National College of Ireland

Cloud Data Security Improvement Using Cryptographic Steganography by Truly Random and Cryptographically Secure Random Number

MSc Research Project
MSc in Cloud Computing

Rajendra Chavan
Student ID: 21124213

School of Computing
National College of Ireland

Supervisor: Prof. Rashid Mijumbi

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Rajendra Anil Chavan
Student ID:	21124213
Programme:	MSc. In Cloud Computing **Year:** 2022-2023
Module:	Research Project
Supervisor:	Prof. Rashid Mijumbi
Submission Due Date:	18th September 2023
Project Title:	Cloud Data Security Improvement Using Cryptographic Steganography by Truly Random and Cryptographically Secure Random Number
Word Count:	5719 **Page Count:** 19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Rajendra Anil Chavan

Date: 14th August 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□

Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Cloud Data Security Improvement Using Cryptographic Steganography by Truly Random and Cryptographically Secure Random Number

Rajendra Chavan
21124213

Abstract

Cloud computing is one of the most significant advances in information technology, but data storage security is a major issue in the cloud environment. The need for security and privacy in the age of digital communication is increasing as more devices are being connected to the internet every day. It is necessary to secure confidential communications during transfer over insecure online channels. Therefore, data security becomes an essential area of research during this era of cloud storage and computing. Steganography is a method for concealing information within an unobservable medium, such as images, text, audio or video files and the topic of this research. The proposed method conceals data such as text within an image using least significant bit (LSB) embedding assisted by a cryptographically secure pseudo-random number (CSPRNG) generator to calculate the pixel coordinates for storing the message bits. The secure key used was generated by derivative of the 1M iterations of a pseudo random function, PBKDF2-HMA-256 on the input password. The iterative pair of values generated from the Chacha20 stream cipher from the secure key that corresponds to the CSPRNG value was used to determine the pixel co-ordinates of the message bits to be hidden. The proposed research approach was implemented and evaluated on different cover images and message inputs. The whole code implementation was done using python and both the encryption and decryption process of the stego image was performed and verified for data integrity. This implementation provides extremely hard to decipher and fool-proof technique in the absence of truly random number generators (TRNGs).

1 Introduction

Since data security has been a topic of concern for some time, it has become a major concern for businesses and individuals that deal with sensitive information. In recent years, cloud computing has become pervasive; it offers several advantages, including cost-effectiveness, scalability, and ease of use; and it has evolved into an efficient method for storing and gaining access to data. There has been an increase in data security issues such as data breaches, larceny, and unauthorized access due to the increased use of cloud computing. This is especially concerning for businesses that manage sensitive data such as financial and medical information. The use of cryptography to protect data in cloud computing is widespread, but it

has limitations (Mustafa et al.; 2018). Attackers can compromise cryptographic keys and exploit weaknesses in encryption techniques.

In contrast, steganography can be used to enhance data security by concealing information in plain sight. Steganography is a technique for concealing information that centres on embedding data within a medium like an image, video or audio. This method can ensure data confidentiality when transmitting sensitive information. However, if the assailant is aware of the information hiding scheme's pattern, data confidentiality is not maintained, unlike with encryption, as the attacker may obtain the data from the storage media, which is typically presented in plaintext form. Therefore, cryptographic steganography is the optimal technique for supporting a high level of data security (Abdullah and Aziz; 2016). In this method, the payload data is encrypted prior to its encrypted result or embedding the ciphertext within the cover image. Only users who possess the secret key can extract and decrypt the ciphertext. In this research, the use of cryptographic steganography employing truly random and cryptographically secure random numbers to enhance the security of cloud data is investigated.

Steganographic and cryptography procedures have both been shown to be effective and reliable. Unfortunately, steganography and cryptography alone have been shown to be inadequate for comprehensive security. Steganography only supports confidentiality and identification security principles, whereas cryptography also supports data integrity and non-repudiation security principles. As a result, several researchers recommend combining the two to create a crypto-steganographic system, which is a stronger and more trustworthy method (Aung and Naing; 2014; Rahmani et al.; 2014). Systems that use crypto steganography offer a lot of potential to improve information security.

1.1 Motivation

Multiple factors motivate the investigation of increasing cloud data security through the use of cryptographic steganography with random and cryptographically secure random numbers. As previously mentioned, the increasing use of cloud computing has resulted in a proportional rise in data security vulnerabilities. Organizations must take precautions to protect their data from unauthorized access, theft, and data intrusions. Steganography has been extensively utilized in a variety of applications, including multimedia and text data. However, there is limited research on the application of steganography to cloud data security. The prospective benefits of using steganography to improve cloud data security have yet to be exhaustively investigated. Furthermore, the use of cryptographically secure random numbers in steganography has the potential to enhance data security by rendering steganographic data less detectable. Studying the secure data transmission of stego images encrypted using CSPRNG for cloud data security is therefore a critical area of study.

1.2 Research Problem

Encryption alone has proven insufficient for providing data security and sustaining data privacy in the cloud computing environment during data storage and transmission. The issue is of general interest and significant significance because it affects a large number of people on an

individual, business, and organizational level, as well as many industries, particularly the IT industry and academia. As a result, the effects of data leakage caused by cyberattacks and surveillance could be detrimental to organizations that use cloud computing services. The effects include data loss and the disclosure of sensitive and confidential information, which can lead to a loss of client confidence and substantial financial setbacks (Adee and Mouratidis; 2022). From a practical standpoint, the current state of cryptography and steganalysis presents numerous obstacles. Numerous experts concur that cutting-edge steganalysis cannot be utilized effectively in the actual world. This issue stems from the difficulty of modeling statistical solutions in a practical setting, where the parameters of theoretical implementations struggle to account for the highly variable aspects of the real world. This research has tried to address these issues by providing a novel approach capable of providing a secure communication in the cloud.

1.3 Research Question

Q: How does the proposed method improve the secure transmission of data on the cloud using CSPRNG?

1.4 Research Objective

The primary objective of this research is to implement a data security mechanism that offers better protection and safeguarding of the sensitive information with the use of CSPNRG for secure key generation and using the key to perform steganography on the user message. A decryption mechanism was also implemented to recover the original message with a password used for encryption.

1.5 Research Contribution

The possible implications of the proposed research are significant, as it aims to improve cloud data security through cryptographic steganography utilizing cryptographically secure random numbers. This endeavour will contribute to the body of knowledge in the field of research by proposing an innovative method for enhancing cloud data security.

- A significant contribution of this study is the development of a novel method for enhancing data security via CSPRNGs.
- The use of CSPRNGs in steganography has the potential to enhance data security by reducing the detectability of steganographic data. This initiative will investigate the application of these numbers in steganography and assess their effectiveness in enhancing cloud data security.
- With the use of PBKDF2-HMA-256 for secure key generation and Chacha20 stream ciphers to generate CSPRNGs for image steganography, the security mechanism of the proposed technique is improved.
- The proposed approach is evaluated for performance in terms of its simplicity of implementation, sophistication of the cryptographic process and the successful encryption and decryption of the secret message with higher reliability of recovery.

2 Related Work

Numerous studies advocates integrating the two techniques, cryptography and steganography into a single framework. Combining the two techniques, as suggested by researchers today, can result in a higher level of security. This literature section provides insight into the existing techniques in both cryptography and steganography and strives to discuss and analyze the proposing a radical approach for the combined implementation of such an approach.

2.1 Cryptography Techniques

According to (Gollmann; 2009), cryptography is a collection of techniques involving the use of decryption and encryption schemes, hash functions, and digital signature models. Encryption systems are used to transform secret messages into formats that are unreadable to unauthorized parties, whereas decryption systems are implemented to decode the encrypted message by an authorized party. Varol et al. (2017) conducted research on symmetric encryption, which is used to encrypt specific texts or speech. In this study, the content which needs to be secured is first transformed into an encapsulating cipher that a cipher algorithm cannot decode. In a study on secure sharing in cloud computing, (Chachapara et al.; 2013) presented an architecture that uses cryptographic techniques like RSA and AES, with AES as regarded as the most reliable algorithm in the field of study. Users of the cloud can create keys for various users with various access rights to their content.

Elharrouss et al. (2020) classified traditional cryptography techniques as encryption and decryption techniques that have been in use for a very long time and are frequently based on the mathematical concepts of substitution and transposition. Katz and Lindell. (2020) identified the current cryptography algorithms as RSA, AES, and elliptical curve cryptography (ECC). These methods are widely used in modern communication networks and electronic transactions, and they provide a high level of security for sensitive data. According to (Yahaya and Ajibola; 2019), current encryption techniques are significantly more secure than their forebears due to the use of sophisticated algorithms and asymmetric keys. In circumstances where security requirements are not as stringent or where simplicity and efficacy are more essential, however, conventional approaches are still applicable.

Two secret keys were introduced as a secure approach in the study by (Fadul and Ahmed; 2013), where the second key would be utilized for both encryption and decryption. The experiment indicated that doing so will strengthen security while maintaining performance metrics close to those of the original AES. A modification to AES that offers safeguarding data on the cloud employing a distinct key generation process along with the addition of a transpose matrix to generate ciphertexts concealed from the eyes of third-party has also been discussed in (Forhad et al.; 2018), which further creates safety to access sensitive data over the cloud. The authors of (Aurora et al.; 2013) addressed the security problems associated with cloud storage. The challenges of providing security for the provider of third-party services are also discussed within the context of the proposed study of various algorithms. Their findings indicate that AES encrypts data in less time than RSA, while RSA requires more time to execute. They reached the conclusion that AES provides superior efficacy overall.

Using the random oracle model, (Percival; 2009) provided a family of key derivation functions that are provably sequential memory-hard for cryptographic hash functions like PBKDF2-HMA-256. Choi and Seo. (2020) integrated several redundant operations by optimizing the internal mechanism of the pseudo random function (PRF) used in PBKDF2. It was done using two HMAC algorithms, SHA-2 and LSH family as the PRF of PBKDF2.

2.2 Steganography Approaches

Two-step steganography technique was proposed by the authors in (Abdulla et al.; 2014). The pre-processing algorithm that reduces the size of the secret images is the initial phase. In the second stage, an algorithm based on the Fibonacci representation of pixel intensity was used as an embedding mechanism. Their findings demonstrated the efficacy of their method against RS and steganalysis WS attacks. However, the proposed method did not protect the confidentiality of sensitive data using encryption techniques. For steganography in (Rahman et al.; 2018), the authors employed the Blowfish encryption algorithm and Least Significant Bits (E-LSB). In addition, the SHA-256 hashing algorithm was used to verify the data's integrity and bolster cloud storage security. According to their findings, a good PSNR value was attained when concealing 1KB of data as an image. The authors in (Hosam and Ahmad; 2019) suggested encrypting cloud-based data with the AES algorithm and a secret key. Then, the confidential encryption key indexes the user's image using the ECC algorithm. Thus, the issue of key management is resolved.

Durgadevi et al. (2019) presented a new secure communication model that combines cryptography and steganography techniques to establish two layers of security so that a steganalyst cannot decipher the ciphertext without the secret key. The key images were encrypted using the Secure Force-AES algorithm before being concealed in the cowl image using the JSTEG and LSB techniques. Encrypted secret data was concealed in a cover image using JSteg and LSB coding for steganography, ensuring exceedingly secure communication. Negi et al. (2021) proposed a hybrid technique that combines AES and Coverless image stenography. In the cryptography portion, after encrypting 128-bit plaintext with a 128-bit key using AES, the encrypted message is converted to the bitstream, and the OMR sheet is bubbled by marking TRUE if the bit in the bitstream is 1 and FALSE otherwise. In this system, a 128-question OMR sheet is used, signifying that OMR contains 128 entries for storing 0/1 bits. Consequently, the OMR can store up to 128 bits of data. The sender delivers the mapped bubble sheet and the secret key used in the mapped sheet to the recipient, who then decodes the message. This message is then decrypted using the obtained key, resulting in the encryption of plaintext. The recipient reads the material received.

Combining stenography with the LSB algorithm and hybrid encryption with RSA and AES, (Mohd Nai and Abdul Aziz; 2021) presented a web application for protecting data by utilizing LSB and Hybrid Encryption with RSA and AES. The RSA technique is employed as an additional layer of security, with the private keys of the intended target being the only means to decrypt it. RSA is used to generate the encryption keys and encrypt the text contents. Utilizing the LSB algorithm, the private key and encrypted message are embedded within the image. The user will embed a message within the image of their choosing and upload it to Google Drive. The recipient can then download the image and extract the message back into

plain text. LSB substitution was utilized for storing information in the random bit position of a randomized pixel location of the cover picture using PRNG in (Ehsan et al.; 2021) image steganography approach. The suggested technique employed a 3-3-2 strategy to conceal a byte within a pixel of a 24-bit color picture.

2.3 Cryptographically secure random number generators (CSPRNGs)

Kumar and Mishra. (2022) evaluated the efficacy of different CSPRNGs against quantum attacks using lattice-based cryptography as a potential solution for generating secure PRNGs with the objective to evaluate various types of RNGs against quantum attacks and ascertain the efficacy of lattice-based PRNGs for post-quantum cryptography. In order to create an initial seed for PRNG and CSPRNG, (Ryan et al.; 2022) used Grammatical Evolution (GE). They showed how effective GE is as an entropy source and found that the initial seeds have an average entropy value of 7.940560934 for 8-bit entropy, which is close to the ideal value of 8. One popular type of CSPRNGs in version 8 is discrete logarithmic pseudorandom number generators. By swapping word-wise arithmetic operations for bit-wise logical operations in trapdoor and hard-core functions, (Koshiba et al.; 2022) suggested a modification to Gennaro's PRNG and established an equivalent between the particular variation of the discrete logarithm problem and the standard problem. The NIST SP 800-22 suite was used to test the CSPRNG, which passed all randomness tests.

To summarize, this section conclusively covered the previous research work done in cryptography, steganography and random number generation. The literature review was greatly helpful in the finalization of the proposed approach of using CSPRNGs for stego-image encryption and decryption. This research work, its methodology, the specifications of the design and implementation details were covered in detail in the subsequent sections.

3 Research Methodology

The proposed solution to the research challenge to enhance cloud data security using cryptographic steganography involves embedding data within cover data using CSPRNG to create a stego-object. The proposed methodology for achieving secure data transmission on cloud is presented in Figure 1.

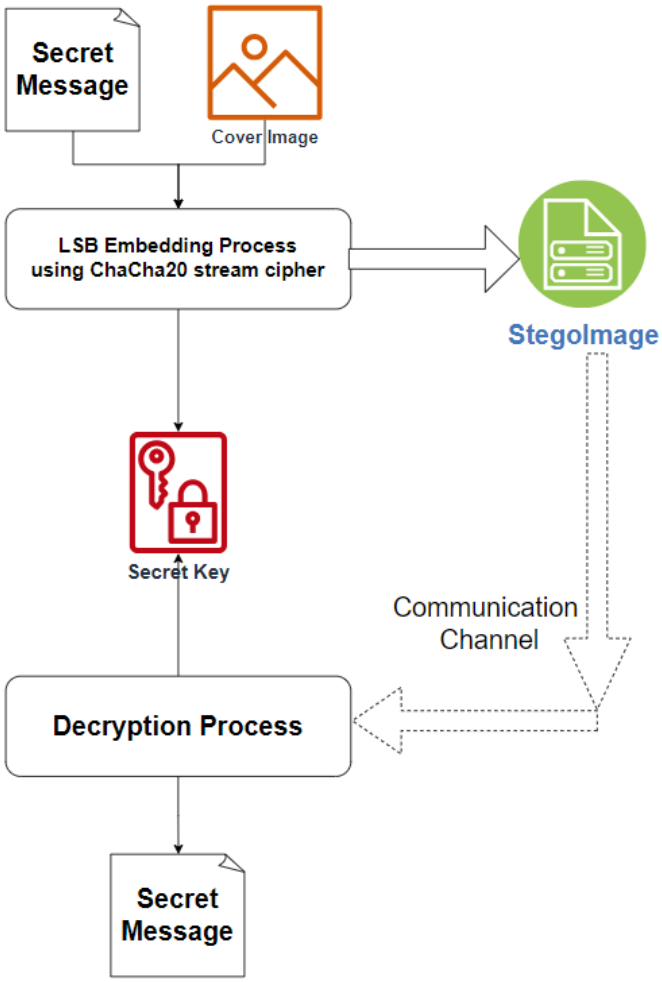

Figure 1. Proposed Research Methodology

The inputs presented to the proposed model are the secret message that needs to be transmitted securely to a remote location on cloud, the password that was used to generate the secret key and the cover image used for hiding the message inside its co-ordinates. The first step in this process was the generation of the secure key. The secure key is generated from the password using password based key derivate function 2 (PBKDF2) using a hash-based message authentication code (HMAC) and SHA256. The generated secure key was then used in the encryption process using Chacha20 stream cipher which is primarily used in file encryption. Most cryptographers those who prefer to secure data without compromising on the performance use it for its speed and security. After the encryption process, the cryptographically encrypted key would be used as the CSPRNG key value for hiding the message in the cover image. This was carried out by generating the *x* and *y* coordinates from the cover image. Next, these coordinates are stored in a list that covers all the coordinates of the cover image. This was done to use the whole cover image to store the message bits. Then, the CSPRNG iterator is used to select the pixel indices of the image to where the secret message bits can be stored. The storage process involves replacing the LSB of each image pixel with the secret message bits until all the bits covered in the LSB of the pixels of the image. This is performed in such a way that it utilized the full coordinate range of the image so that it would be hard to decipher the messages due to the input message size and the large size of the cover

image. Once it was done, the stego image was saved locally, before it can be transmitted over the cloud securely.

The second part of the research methodology was focused on retrieving the message from the stego image securely. This process was comparatively easier to achieve as this only the password and the stego image as the input from the user. If the password verification fails, then the decryption process is terminated. Upon the verification of the password from the system, the stego image was processed by the CSPRNG module that begins the retrieval process by checking if the bits to recover were less than the image capacity. As long as this condition is met, the decryption process runs to completion and the secret message is saved to a new file. The proposed method is significantly fast and does not consume time as much as an AES or RSA based encryption/decryption process.

4 Design Specifications

In this paper, a two-step model for cloud data security was proposed. In the first stage, the encryption, CSPRNG keys are generated using Chacha20 and PBKFD2-HMAC-SHA256 algorithms. LSB embedding was used as the steganography algorithm to hide the data at the pixel locations generated by the CSPRNG key. This secure message to be hidden in the cover image using the combination of both CSPRNG and LSB embedding to create the stego-image. In the next stage, the decryption stage, the stego-image and password are provided as inputs to the CSPRNG module that verifies the password and decrypts the message from the stego-image. The whole process is simple and easily adapted to any cloud data communication and transmission systems.

To elaborate further on the design specification, this section is divided into three sections:

1. Secure encrypted CSPRNG key generation using NumberGenerator.
2. Data-hiding into the cover image using Steganography.
3. Secret message retrieval using decryption process.

Each section is elaborated in sufficient detail to precisely understand the process behind the proposed research approach.

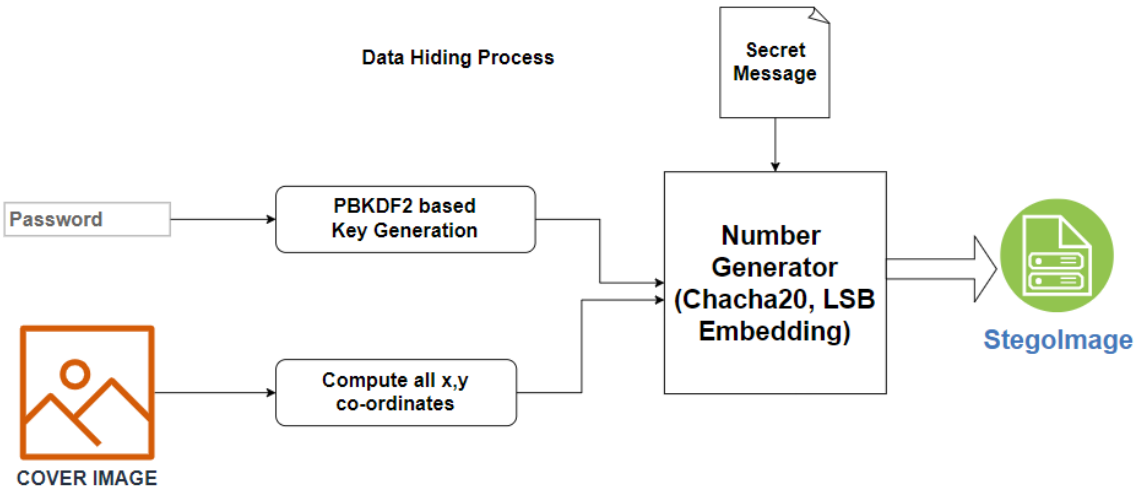

Figure 2. Key generation and Data hiding block

Figure. 2 presents the process flow diagram of the proposed CSPRNG based Steganography technique. This can be further divided into two sections as below.

A. CSPRNG Key Generation

Instead of using a conventional PRNG, CSPRNG should be used for applications that demand a greater level of security. To create safe tokens, perform secure communications, and generate encryption keys, CSPRNGs provide random numbers that are suitable for use in security-critical applications. A CSPRNG is made to look exactly like real random numbers for all intents and purposes. CSPRNGs produce what appears to be a random series of numbers but is really produced using a deterministic algorithm plus an initial random seed value. Usually, the initial seed value comes from physical sources like radioactive decay, ambient noise, or mouse movements.

A CSPRNG's strength is its capacity to produce results that, even to an attacker with in-depth knowledge of the method and seed values used, are statistically indistinguishable from truly random results. This qualifies CSPRNGs for usage in cryptographic applications where the robustness of the random number generator determines the system's security.

In this work, the CSPRNG key generation is presented in Figure. 3 for further understanding. The process of secure key generation is initiated by the password provided at the input. The password is then processed by PBKDF2-HMAC-256, a password-based key derivative function. The user provided password is hashed with SHA-256, a secure hashing algorithm to map it to a fixed length of 32 bytes. The output is then used to initialize the *PBKDF2-HMAC,* a hash-based message authentication code which uses a hash and secret key. In this work, the use of HMAC was suggested because of its functionality of achieving

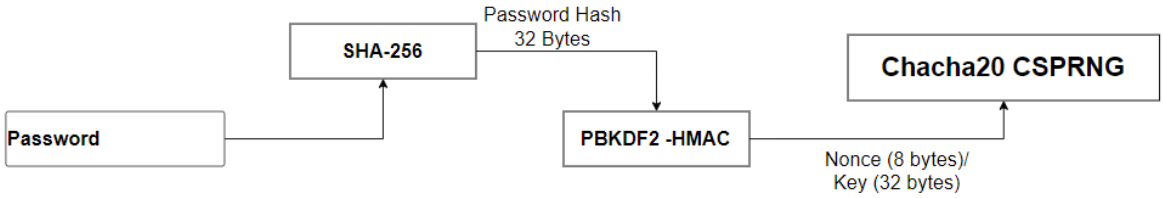

Figure 3. CSPRNG Key Generation

authentication and verify that the data is authentic with shared secrets in contrast to other algorithms that use signatures and asymmetric cryptography processes. PBKDF2-HMAC splits the output from SHA-256 into two 16-bit blocks that was used as key and salt. The *salt* was used together with hashing to make it even more secure. The secure key is derived from PBKDF2-HMAC-SHA256 with 1M iterations. Later, with the generated secure key, a number generator and *Chacha20* are initialized. The choice of *Chacha20* is due to its robust cryptographic characteristics and high-speed encryption, including as resistance to a variety of attacks like differential and linear cryptanalysis. Although ChaCha20 has not received as much research attention as some other algorithms, such as the AES, it has undergone extensive examination and is regarded as secure for many real-world use situations. Chacha20 was used as the PRNG, with the key material used as 8 bytes *nonce*, a mandatory value that would never be reused for any other encryption process done using this generated key and 32 bytes key for the cipher stream.

Apart from this, another less secure option is provided for those who do not prefer to use password for the fear of losing it. In such as case, the random numbers were generated using the python library, '*secrets'* for generating cryptographically strong random numbers suitable for managing data such as passwords, account authentication, security tokens, and related secrets. The next step in the design process is the use of LSB embedding for steganography and the role of CSPRNG key in securing the message hiding mechanism.

B. *LSB STEGANOGRAPHY FOR DATA-HIDING*

The LSB algorithm is the simplest and most commonly used steganography algorithm. It is based on embedding the bits of the secret text message into at least three significant bits of the cover image pixels. The confidential text message bits were concealed in the position of the least significant bit of the individual pixels of the cover image. In the LSB algorithm, each bit of the data to be concealed is appended to the last bit of a byte of the data that generates the cover image. In the proposal, 24-bit images were used because each pixel contained three bits of information, one for each layer of RGB colors in the cover image. Figure 4 shows the LSB embedding process for the three RGB layers of the image.

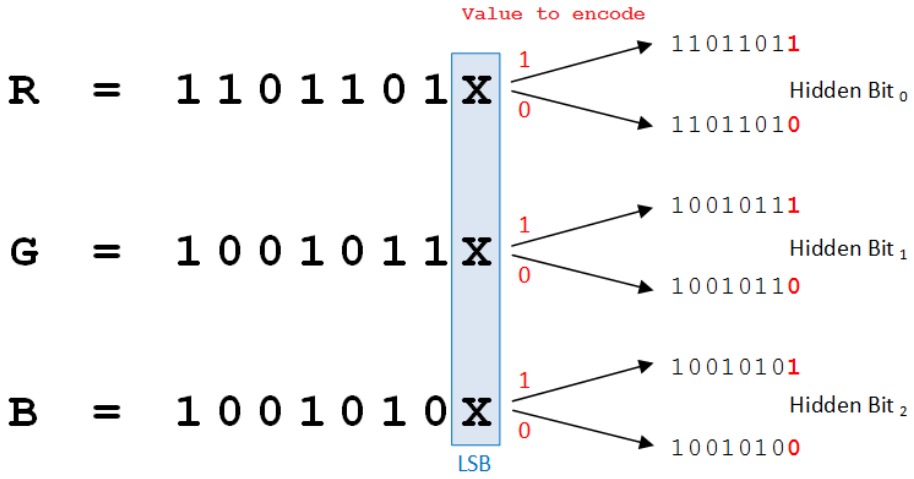

Figure 4. Secret data hiding using LSB in RGB color cover image

LSB was applied to the cover image using confidential coordinates that was acquired from the CSPRNG key value. The CSPRNG result was used to determine the index in the list of coordinates that will be used for storing the LSB bits and generate the stego image. Any authorized receiver must possess the cover image coordinates and the bits order into color channels in order to decipher the confidential message.

C. SECURE IMAGE DECRYPTION

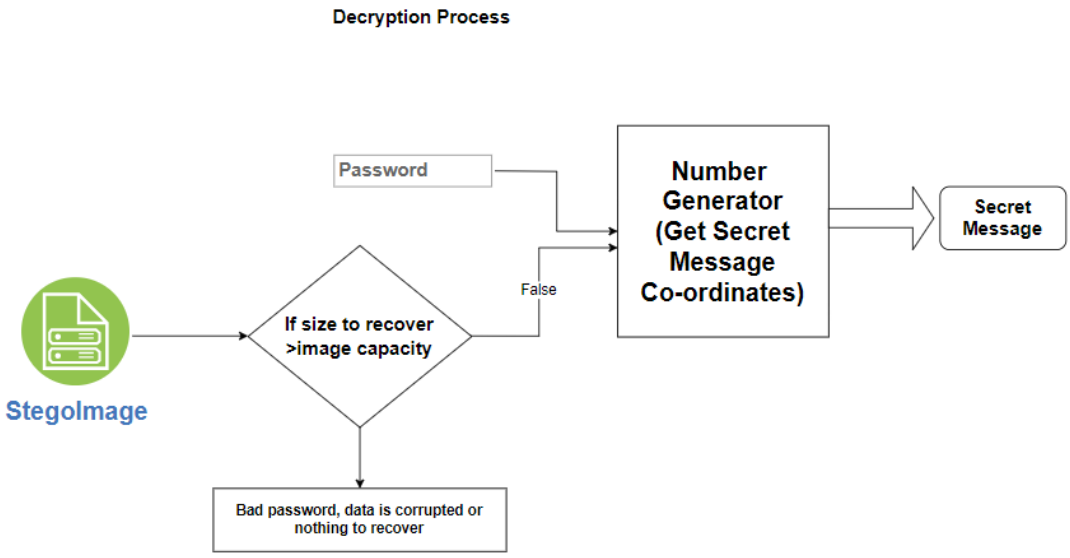

Figure 5. Cover image decryption process to recover secret message.

The cover image decryption process shown in Figure 5 is relatively straight forward in relation to the sophistication behind the key generation process and LSB embedding for stego image generation. The decryption block takes the password and the stego image as the input. If the password verification is unsuccessful, the decryption operation is stopped. The CSPRNG

module processed the stego image after the system verified the password, and it started the recovery process by determining whether the volume of data required to be recovered was smaller than the image's storage capacity. The secret message is saved to another file and the decryption procedure is completed as long as this requirement is satisfied.

5 Implementation

This section presents the proposed CSPRNG based Steganography model implementation using Python programming language and Pycryptodome, an extensive Python package that offers a variety of cryptography protocols and methods. It's used in Python programs to accomplish several cryptographic operations like encryption, decryption, hashing, and digital signatures. A user-friendly and secure approach to deal with cryptography in Python is what PyCryptodome promises to provide. The implementation using python will be explained using the python code blocks involved in the coding process behind CSPRNG encryption and decryption.

```
import argparse
from PIL import Image, UnidentifiedImageError
import os
import sys
import hashlib
import secrets
from Crypto.Cipher import ChaCha20
from math import ceil

VERSION = 2.0
ENCODED_SIZE_BITS_LENGTH = 32
```

Figure 6. Import required libraries in python.

Crypto.Cipher is the pycryptodome python package where many cryptographic algorithms including *chacha20* are present as shown in Figure 6.

```
def _gen_key(self, password=None) -> bytes:

    if password:
        key = hashlib.pbkdf2_hmac("sha256", bytes(
            password, "utf-8"), bytes(), 1000000)
    else:
        key = secrets.randbits(256).to_bytes(32, sys.byteorder)

    return key
```

Figure 7. Key generation using PBKDF2-HMAC.

Figure 7. shows the key generation block using PBKDF2 iterated over 1 million times for additional security. *Secrets.randbits* can be used to generate random key of size 256 bytes if no password is provided as input to the hash function.

```
class NumberGenerator():

    def __init__(self, x: int, y: int, key: bytes):
        if not (x and y and key):
            raise NumberGeneratorException("Bad coordinates or key")

        self.ctr = bytes(1)
        self.coord = self._generate_coord(x, y)
        self.prng = ChaCha20.new(key=key, nonce=bytes(8))
        #print (prng)
```

Figure 8. NumberGenerator Class

The NumberGenerator Class is used to create the CSPRNG key using *chacha20* stream cipher as in Figure 8. It also calls a routine, *_generate_coord(x,y)* to store all the x, y coordinates of the image to a list.

```
    def hide(self, data_file: str, output_file: str) -> None:

        file_len = os.path.getsize(data_file)
        bits_to_hide_len = file_len * 8

        if (bits_to_hide_len > self.image_capacity):
            raise SteganoIMGOverflowError()

        data = self._get_data_from_file(data_file)

        x, y = 0, 0
        rgb = 0
        tmp_bits = [bin(t)[2:] for t in data]
        bits_to_hide = ["0" * (8 - len(t)) + t if len(t)
                        < 8 else t for t in tmp_bits]

        # Total length is written on the ENCODED_SIZE_BITS_LENGTH first bits
        binary_encoded_size = "0" * \
            (ENCODED_SIZE_BITS_LENGTH -
             len(bin(bits_to_hide_len)[2:])) + bin(bits_to_hide_len)[2:]

        bits_to_hide.insert(0, binary_encoded_size)
```

Figure 9. Hide routine to perform LSB embedding

Figure 9 shows the *hide()* routine performs LSB embedding on to the lsb bit of the pixel coordinates whose indices are selected by CSPRNG. This iterative process creates the stego-

```
def recover(self, output_file: str) -> None:

    while (ctr < ENCODED_SIZE_BITS_LENGTH):
        if rgb == 0:
            try:
                x, y = next(self.numberGenerator)
            except StopIteration:
                raise SteganoIMGException("No more coordinates to use")

        # Left bit shifft and add the bit from LSB of colour pixel
        size_to_recover = (size_to_recover << 1) + \
            (self.img[x, y][rgb] & 1)
        rgb = (rgb + 1) % 3
        ctr += 1
```

image.

Figure 10. Secret message retrieval based on a rule

6 Evaluation

The results are evaluated by the way of executing the python program to perform encryption to create the stego image and decryption to recover the secret message. The initial directory structure of the code folder is shown below.

```
Directory of C:\Users\USER\Documents\Rajendra\CryptSteg

12-08-2023  02:08    <DIR>          .
11-08-2023  22:54    <DIR>          ..
07-08-2023  15:24               120 commands.txt
12-02-2017  17:29         4,953,978 cover.png
12-08-2023  00:24             9,735 csprng-steg.py
12-08-2023  01:53    <DIR>          images
06-08-2023  21:48               360 mydata.txt
13-09-2020  18:45                19 requirements.txt
               5 File(s)      4,964,212 bytes
               3 Dir(s)  42,445,586,432 bytes free
```

Figure 11. Code source directory

Figure 11 depicts the source directory of the code, *csprng-steg.py*, with the cover image, *cover.png* and secret message stored in *mydata.txt*. Note the sizes of these files, before and after the steganography process.

To run the experiment, the python script has to be invoked with the appropriate arguments. The python script takes the following arguments for encryption as in Figure 12:

- Choice – *hide* – for encryption
- Cover image as a *.png* file
- Password text, can contain alphanumerical characters
- Input message to be embedded into the cover image.

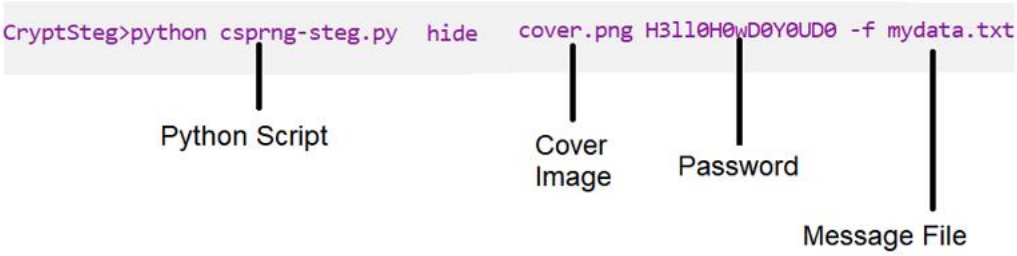

Figure 12: Hide Message - Encrypt

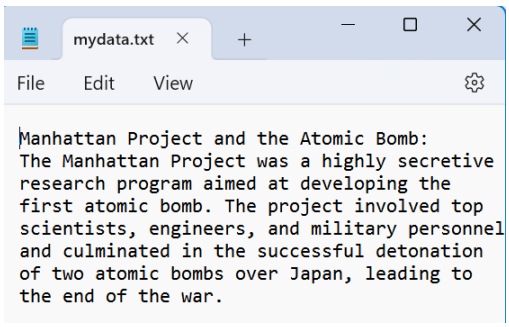

Figure 13: Secret Message

On executing the script for encryption,

```
(cryptosteg) C:\Users\USER\Documents\Rajendra\CSPRNG_STEG>python csprng-steg.py hide cover.png H3ll0H0wD0Y0UD0 -f mydata.txt
PBKDF2-HMAC-SHA256 function used to generate secure key
Stego-File Name:  Cover_Steg.png


Message Hidden... Successfully Completed Operations..

Check Folder for the output stegoimage
StegoImage Generated in --- 2.303426504135132 seconds ---
```

Figure 14: CSPRNG-Steganography Output

```
12-08-2023  02:48    <DIR>          .
11-08-2023  22:54    <DIR>          ..
07-08-2023  15:24               120 commands.txt
12-02-2017  17:29         4,953,978 cover.png
12-08-2023  02:48         4,942,434 Cover_Steg.png
12-08-2023  02:47             9,945 csprng-steg.py
12-08-2023  01:53    <DIR>          images
12-08-2023  02:20               323 mydata.txt
13-09-2020  18:45                19 requirements.txt
               6 File(s)      9,906,819 bytes
               3 Dir(s)  42,375,413,760 bytes free
```

Figure 15: Directory tree after performing steganography

Figure. 14 provides the system output for the stego-image generation and the time taken for generating the stego-image, from secure key generation using PBKDF2, encryption using chacha20 and finally, using LSB embedding to store the secure image within the cover image, took only 2.303 seconds. This is significantly lesser than other AES or RSA based encryption methods. Though the encryption time is dependent on the key length, cover image size etc., it can be confidently said that the proposed approach performs stego-image generation with more security in a relatively faster time on a standard computing PC platform. Now, on observing the sizes of the cover image and stego-image, it can be realised that stego-image is a little lesser than the cover image. This is due to the fact that in most cases, the quantity of concealed data (information) that can be contained within a stego image is not as large as the total number of pixels in the cover image. Figure 16 shows the python command line script for the decryption process.

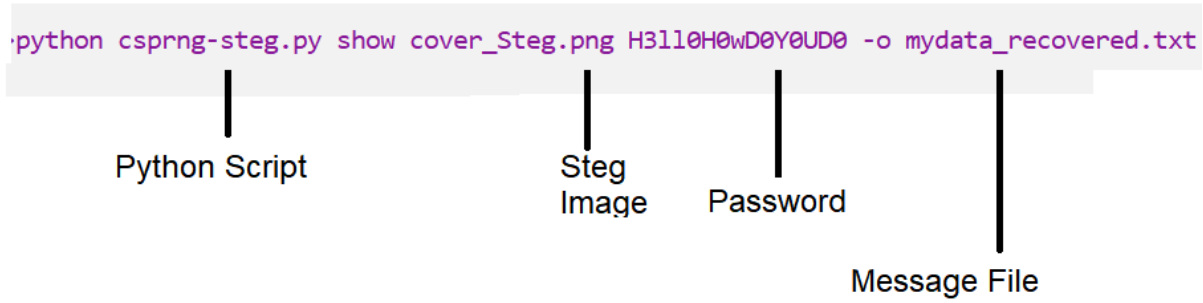

Figure 16: Recover Message – Decrypt

On executing the script for decryption,

```
(cryptosteg) C:\Users\USER\Documents\Rajendra\CSPRNG_STEG>python csprng-steg.py show cover_Steg.png H3ll0H0wD0Y0UD0 -o mydata_rec.txt
PBKDF2-HMAC-SHA256 function used to generate secure key
Secret Message Recovered Successfully!!
Secret Message Decrypted in --- 1.5160458087921143 seconds ---
```

Figure 17: Decryption Output

From Figure 17, it was observed that the decryption process took just 1.516 seconds to complete, better than other conventional AES/RSA based encryption processes. Figure 18 shows the recovered secret message.

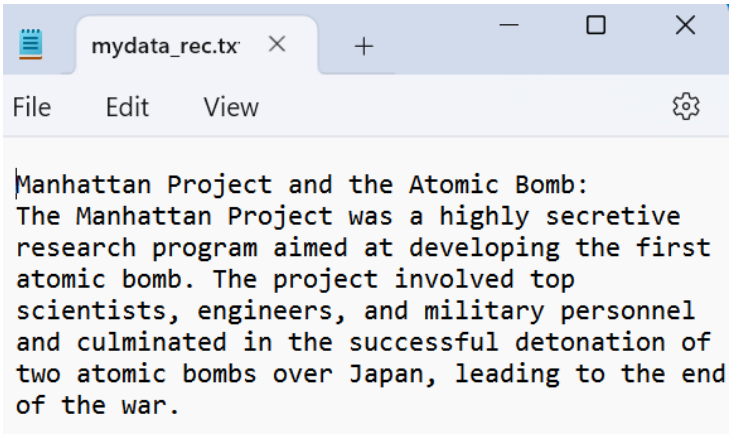

Figure 18: Recovered - Secret Message

```
 Directory of C:\Users\USER\Documents\Rajendra\CryptSteg

12-08-2023  02:55    <DIR>          .
11-08-2023  22:54    <DIR>          ..
07-08-2023  15:24               120 commands.txt
12-02-2017  17:29         4,953,978 cover.png
12-08-2023  02:48         4,942,434 Cover_Steg.png
12-08-2023  02:47             9,945 csprng-steg.py
12-08-2023  01:53    <DIR>          images
12-08-2023  02:20               323 mydata.txt
12-08-2023  02:55               323 mydata_recovered.txt
13-09-2020  18:45                19 requirements.txt
               7 File(s)      9,907,142 bytes
               3 Dir(s)  42,344,308,736 bytes free
```

Figure 19: Directory Tree with both the mydata.txt and mydata_recovered.txt shown with the same file size, confirming successful recovery of the secret message.

This research work was built upon the work done by (Revanna; 2021) where PRNGs were used to determine the locations of the pixels of the cover image to store the encoded pixel values of another secret image. They had used evaluation metrics like peak signal-to-noise ratio (PSNR) and mean-squared error (MSE), specific to work with images which cannot be carried over here. But the stego-image generation method used in the research is more complex and secure than their PRNG based approach.

7 Conclusion

In conclusion, a strong and adaptable method for secret data transfer is produced by integrating a CSPRNG utilizing the ChaCha20 algorithm. ChaCha20, which is renowned for its high security features and effective performance, provides a reliable framework for encrypting sensitive data and embedding it into digital images. Because the generated numbers are pseudorandom, using a ChaCha20-based CSPRNG for steganography ensures that the hidden

data is secure and resistant to attempts at cryptanalysis and statistical analysis. Additionally, ChaCha20's speed and efficiency enable easy integration, making it appropriate for real-time applications and scenarios with limited resources. This study was initially set to work with the generation of truly random number generators (TRNGs) too along with CSPRNGs. But generating randomness using software- based algorithms are deterministic by nature, i.e., though the generated numbers are statistically random, they are, however, decided by an initial seed value. So, using hardware-based generators was the only alternative option available, which could not be considered due to limited resources. The future research direction can consider the scope of investigating the application of CSPRNGs and ChaCha20 in diverse steganography elements, such as audio, video, or 3D models.

References

Abdullah, A.M. and Aziz, R.H.H., 2016. New approaches to encrypt and decrypt data in image using cryptography and steganography algorithm. *International Journal of Computer Applications*, *143*(4), pp.11-17.

Adee, R. and Mouratidis, H., 2022. A dynamic four-step data security model for data in cloud computing based on cryptography and steganography. *Sensors*, *22*(3), p.1109.

Arora, R., Parashar, A. and Transforming, C.C.I., 2013. Secure user data in cloud computing using encryption algorithms. *International journal of engineering research and applications*, *3*(4), pp.1922-1926.

Aung, P.P. and Naing, T.M., 2014. A novel secure combination technique of steganography and cryptography. *International Journal of Information Technology, Modeling and Computing (IJITMC)*, *2*(1), pp.55-62.

Chachapara, K. and Bhadlawala, S., 2013, November. Secure sharing with cryptography in cloud computing. In *2013 Nirma University International Conference on Engineering (NUiCONE)* (pp. 1-3). IEEE.

Choi, H. and Seo, S.C., 2020, August. Optimization of PBKDF2-HMAC-SHA256 and PBKDF2-HMAC-LSH256 in CPU Environments. In *International Conference on Information Security Applications* (pp. 321-333). Cham: Springer International Publishing.

Durgadevi, S., Jayasrilakshmi, S., Mohanraj, S. and Anbarasi, M.S., 2019. Enhance security for medical images through secure force cryptography with steganography techniques. *Int Res J Eng Technol*, *6*.

Elharrouss, O., Almaadeed, N. and Al-Maadeed, S., 2020, February. An image steganography approach based on k-least significant bits (k-LSB). In *2020 IEEE international conference on informatics, IoT, and enabling technologies (ICIoT)* (pp. 131-135). IEEE.

Fadul, I.M.A. and Ahmed, T.M.H., 2013. Enhanced security of Rijndael algorithm using two secret keys. *International Journal of Security and its applications*, *7*(4), pp.127-134.

Forhad, M.S.A., Riaz, S., Hossain, M.S. and Das, M., 2018. An improvement of advanced encryption standard. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, *18*(11), pp.159-166.

Gollman, D., 2009. *Computer Securuty*. John Wiley & Sons.

Hosam, O. and Ahmad, M.H., 2019. Hybrid design for cloud data security using combination of AES, ECC and LSB steganography. *International Journal of Computational Science and Engineering*, *19*(2), pp.153-161.

Heeger, J., Yannikos, Y. and Steinebach, M., 2022. An Introduction to the exFAT File System and How to Hide Data Within. *Journal of Cyber Security and Mobility*, pp.239-264.

Katz, J. and Lindell, Y., 2020. *Introduction to modern cryptography*. CRC press.

Kumar, A. and Mishra, A., 2022, April. Evaluation of Cryptographically Secure Pseudo Random Number Generators for Post Quantum Era. In *2022 IEEE 7th International conference for Convergence in Technology (I2CT)* (pp. 1-5). IEEE.

Mohd Nai, M.K.A. and Abdul Aziz, S.R., 2021. Web-Application for securing message using Steganography with LSB Algorithm and Hybrid Encryption.

Negi, L. and Negi, L., 2021, July. Hybrid approach for Data Security using Coverless Image Steganography with AES. In *2021 6th International Conference on Communication and Electronics Systems (ICCES)* (pp. 1077-1083). IEEE.

Percival, C., 2009. Stronger key derivation via sequential memory-hard functions.

Rahman, M.O., Hossen, M.K., Morsad, M.G. and Chandra, A., 2018. An approach for enhancing security of cloud data using cryptography and steganography with e-lsb encoding. *IJCSNS*, *18*(9), p.85.

Rahmani, M.K.I., Arora, K. and Pal, N., 2014. A crypto-steganography: A survey. *International Journal of Advanced computer science and applications*, *5*(7).
Forhad, M.S.A., Riaz, S., Hossain, M.S. and Das, M., 2018. An improvement of advanced encryption standard. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, *18*(11), pp.159-166.

Revanna Kumar, P., 2021. *Cloud Data Security Improvement Using Steganography by Pseudo Random Number Generation (PRNG)* (Doctoral dissertation, Dublin, National College of Ireland)

Varol, N., Aydogan, A.F. and Varol, A., 2017, April. Cyber-attacks targeting android cellphones. In *2017 5th International Symposium on Digital Forensic and Security (ISDFS)* (pp. 1-5). IEEE.

Yahaya, M.M. and Ajibola, A., 2019. Cryptosystem for secure data transmission using advance encryption standard (AES) and steganography. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT)*, *5*(6), pp.317-322.