A Mini Project Report

*on*

**Multi-variate data Analyses and Prediction**

In Subject: Multivariate Analysis

*by*

Prathamesh Lawand : 371031

Om suryawanshi : 371071

Abhishek Walunj : 371065



Faculty Guidance: Prof. Prranjali Jadhav

Head of Department: Dr.Parikshit Mahalle

Department of Artificial Intelligence and Data Science

VIIT

**2022-2023**

**Contents**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Topic** | | **Page No.** |
| **Chapter-1** | **Introduction** | |  |
|  | 1.1 | Introduction | 3 |
|  | 1.2 | Requirements | 5 |
|  | 1.3 | Problem Statement | 5 |
|  | 1.4 | Motivation | 6 |
| **Chapter-2** | **Methodology** | |  |
|  | 2.1 | Purpose | 7 |
|  | 2.2 | Approach | 8 |
|  | 2.3 | Program Code | 10 |
|  | 2.4 | Outcome | 37 |
| **Chapter-3** | **Conclusion** | | 38 |
|  | **References** | | 39 |

**Introduction**

**1.1 Intro:**

Security attacks against network are increasing significantly with time. Our communication media should also be secure and confidential. Cryptanalysis is the study used to describe the methods of code-breaking or cracking the code without using the security information, usually used by hackers. For this purpose, these three suggestions arrive in every one’s mind: (i) one can transmit the message secretly, so that it can be saved from hackers, (ii) the sender ensures that the message arrives to the desired destination, and (iii) the receiver ensures that the received message is in its original form and coming from the authenticate person. In order to achieve the same one can use two techniques, (i) one can use invisible ink for writing the message or can send the message through the confidential person, and (ii) use of scientific approach called ―Cryptography‖. Cryptography is the technique used to avoid unauthorized access of data. Data can be encrypted using a cryptographic algorithm by various keys. It will be transmitted in an encrypted state, and later decrypted by the intended party. If a third party intercepts the encrypted data, it will be difficult to decipher. The security of modern cryptosystems is not based on the secrecy of the algorithm, but on the secrecy of a relatively small amount of information, called a secret key. The fundamental and classical task of cryptography is to provide confidentiality by encryption methods. It is used in applications present in technologically advanced societies; it includes the security of ATM cards, computer passwords, and electronic commerce. For secured communication between two parties following points are considered:

* Encryption: the process to convert the original message into coded form with the help of key, i.e., plain-text into cipher-text is known as encryption.
* Decryption: the reverse process of encryption, i.e., to convert cipher-text into plain-text with the help of key is known as decryption.

**1.2 Requirement:**

* Python
* Python Modules

**1.3 Problem statement:**

Create a GUI interface for encryption and decryption of any type of file.

**1.4 Motivation**

An encryption algorithm provides Confidentiality, Authentication, Integrity and Non-repudiation. Confidentiality ensures that the information is accessible to only authorized set of people. Authentication is the act of establishing that the algorithm is genuine. Integrity in general means completeness but in encryption it is adhering to some set of principles. It is based on consistency with some mathematical proof. Nonrepudiation in cryptology means that it can be verified that the sender and the recipient were, in fact, the parties who claimed to send or receive the message, respectively. In separate used systems, the computers are exposed to the other users. To keep the data secured from different users various encryption algorithms are entered. As computer systems become more pervasive and complex, security is increasingly important. Cryptographic algorithms and protocols constitute the central component of systems that protect network transmissions and store data. Encryption is the translation of data to a secret code. Apart from its uses in Military and Government to facilitate secret communication, Encryption is used in protecting many kinds of civilian systems such as Internet e-commerce, Mobile networks, automatic teller machine transactions.

**Methodology**

**2.1 Purpose:**

**IP Sec functions by using cryptographic techniques. The term cryptography refers to methods of making data unreadable or undecipherable by anyone except the authorized recipient in the event that the message is intercepted by someone else. IP Sec uses cryptography to provide three basic services:**

* [**Authentication**](https://www.sciencedirect.com/topics/computer-science/authentication)
* **Data integrity**
* [**Data confidentiality**](https://www.sciencedirect.com/topics/computer-science/data-confidentiality)

**There are times when only one or two of these services is needed, and other times when all of these services are needed. We will take a look at each of these services individually.**

**2.2 Approach:**

SHA 256 is a part of the SHA 2 family of algorithms, where SHA stands for Secure Hash Algorithm. Published in 2001, it was a joint effort between the NSA and NIST to introduce a successor to the SHA 1 family, which was slowly losing strength against [brute force attacks.](https://www.simplilearn.com/tutorials/cryptography-tutorial/brute-force-attack)

The significance of the 256 in the name stands for the final hash digest value, i.e. irrespective of the size of plaintext/cleartext, the hash value will always be 256 bits.

The other algorithms in the SHA family are more or less similar to SHA 256. Now, look into knowing a little more about their guidelines.

Some of the standout features of the SHA algorithm are as follows:

* Message Length: The length of the cleartext should be less than 264 bits. The size needs to be in the comparison area to keep the digest as random as possible.
* Digest Length: The length of the hash digest should be 256 bits in SHA 256 algorithm, 512 bits in SHA-512, and so on. Bigger digests usually suggest significantly more calculations at the cost of speed and space.
* Irreversible: By design, all hash functions such as the SHA 256 are irreversible. You should neither get a plaintext when you have the digest beforehand nor should the digest provide its original value when you pass it through the hash function again

**2.3Program Code:**

*import* os

*import* sys

*import* hashlib

*import* tkinter *as* tk

*from* tkinter *import* filedialog

*from* tkinter *import* messagebox

*from* Cryptodome.Cipher *import* AES

class EncryptionTool:

    def \_\_init\_\_(*self*, *user\_file*, *user\_key*, *user\_salt*):

*# get the path to input file*

        self.user\_file = user\_file

        self.input\_file\_size = os.path.getsize(self.user\_file)

        self.chunk\_size = 1024

        self.total\_chunks = (self.input\_file\_size // self.chunk\_size) + 1

*# convert the key and salt to bytes*

        self.user\_key = bytes(user\_key, "utf-8")

        self.user\_salt = bytes(user\_key[::-1], "utf-8")

*# get the file extension*

        self.file\_extension = self.user\_file.split(".")[-1]

*# hash type for hashing key and salt*

        self.hash\_type = "SHA256"

*# encrypted file name*

        self.encrypt\_output\_file = ".".join(self.user\_file.split(".")[:-1]) \

            + "." + self.file\_extension + ".kryp"

*# decrypted file name*

        self.decrypt\_output\_file = self.user\_file[:-5].split(".")

        self.decrypt\_output\_file = ".".join(self.decrypt\_output\_file[:-1]) \

            + "\_\_dekrypted\_\_." + self.decrypt\_output\_file[-1]

*# dictionary to store hashed key and salt*

        self.hashed\_key\_salt = dict()

*# hash key and salt into 16 bit hashes*

        self.hash\_key\_salt()

    def read\_in\_chunks(*self*, *file\_object*, *chunk\_size*=1024):

*while* True:

            data = file\_object.read(chunk\_size)

*if* not data:

*break*

*yield* data

    def encrypt(*self*):

*# create a cipher object*

        cipher\_object = AES.new(

            self.hashed\_key\_salt["key"],

            AES.MODE\_CFB,

            self.hashed\_key\_salt["salt"]

        )

        self.abort() *# if the output file already exists, remove it first*

        input\_file = open(self.user\_file, "rb")

        output\_file = open(self.encrypt\_output\_file, "ab")

        done\_chunks = 0

*for* piece *in* self.read\_in\_chunks(input\_file, self.chunk\_size):

            encrypted\_content = cipher\_object.encrypt(piece)

            output\_file.write(encrypted\_content)

            done\_chunks += 1

*yield* (done\_chunks / self.total\_chunks) \* 100

        input\_file.close()

        output\_file.close()

*# clean up the cipher object*

*del* cipher\_object

    def decrypt(*self*):

*#  exact same as above function except in reverse*

        cipher\_object = AES.new(

            self.hashed\_key\_salt["key"],

            AES.MODE\_CFB,

            self.hashed\_key\_salt["salt"]

        )

        self.abort() *# if the output file already exists, remove it first*

        input\_file = open(self.user\_file, "rb")

        output\_file = open(self.decrypt\_output\_file, "xb")

        done\_chunks = 0

*for* piece *in* self.read\_in\_chunks(input\_file):

            decrypted\_content = cipher\_object.decrypt(piece)

            output\_file.write(decrypted\_content)

            done\_chunks += 1

*yield* (done\_chunks / self.total\_chunks) \* 100

        input\_file.close()

        output\_file.close()

*# clean up the cipher object*

*del* cipher\_object

    def abort(*self*):

*if* os.path.isfile(self.encrypt\_output\_file):

            os.remove(self.encrypt\_output\_file)

*if* os.path.isfile(self.decrypt\_output\_file):

            os.remove(self.decrypt\_output\_file)

    def hash\_key\_salt(*self*):

*# --- convert key to hash*

*#  create a new hash object*

        hasher = hashlib.new(self.hash\_type)

        hasher.update(self.user\_key)

*# turn the output key hash into 32 bytes (256 bits)*

        self.hashed\_key\_salt["key"] = bytes(hasher.hexdigest()[:32], "utf-8")

*# clean up hash object*

*del* hasher

*# --- convert salt to hash*

*#  create a new hash object*

        hasher = hashlib.new(self.hash\_type)

        hasher.update(self.user\_salt)

*# turn the output salt hash into 16 bytes (128 bits)*

        self.hashed\_key\_salt["salt"] = bytes(hasher.hexdigest()[:16], "utf-8")

*# clean up hash object*

*del* hasher

*# class EncryptionThread(threading.Thread):*

*#     mutual\_space = {}*

*#     threadLock = threading.Lock()*

*#     def \_\_init\_\_(self, index):*

*#         threading.Thread.\_\_init\_\_(self)*

*#         self.threadID = index*

*#     def run(self):*

*#         try:*

*#             pass*

*#         except Exception as e:*

*#             print(e)*

*#             return*

*#         # Get lock to synchronize threads*

*#         self.threadLock.acquire()*

*#         # Append stuff to mutual\_space*

*#         # Free lock to release next thread*

*#         self.threadLock.release()*

class MainWindow:

    """ GUI Wrapper """

*# configure root directory path relative to this file*

    THIS\_FOLDER\_G = ""

*if* getattr(sys, "frozen", False):

*# frozen*

        THIS\_FOLDER\_G = os.path.dirname(sys.executable)

*else*:

*# unfrozen*

        THIS\_FOLDER\_G = os.path.dirname(os.path.realpath(\_\_file\_\_))

    def \_\_init\_\_(*self*, *root*):

        self.root = root

        self.\_cipher = None

        self.\_file\_url = tk.StringVar()

        self.\_secret\_key = tk.StringVar()

        self.\_salt = tk.StringVar()

        self.\_status = tk.StringVar()

        self.\_status.set("---")

        self.should\_cancel = False

        root.title("KrypApp")

        root.configure(*bg*="#eeeeee")

*try*:

            icon\_img = tk.Image(

                "photo",

*file*=self.THIS\_FOLDER\_G + "/assets/icon.png"

            )

            root.call(

                "wm",

                "iconphoto",

                root.\_w,

                icon\_img

            )

*except* Exception:

*pass*

        self.menu\_bar = tk.Menu(

            root,

*bg*="#eeeeee",

*relief*=tk.FLAT

        )

        self.menu\_bar.add\_command(

*label*="How To",

*command*=self.show\_help\_callback

        )

        self.menu\_bar.add\_command(

*label*="Quit!",

*command*=root.quit

        )

        root.configure(

*menu*=self.menu\_bar

        )

        self.file\_entry\_label = tk.Label(

            root,

*text*="Enter File Path Or Click SELECT FILE Button",

*bg*="#eeeeee",

*anchor*=tk.W

        )

        self.file\_entry\_label.grid(

*padx*=12,

*pady*=(8, 0),

*ipadx*=0,

*ipady*=1,

*row*=0,

*column*=0,

*columnspan*=4,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        self.file\_entry = tk.Entry(

            root,

*textvariable*=self.\_file\_url,

*bg*="#fff",

*exportselection*=0,

*relief*=tk.FLAT

        )

        self.file\_entry.grid(

*padx*=15,

*pady*=6,

*ipadx*=8,

*ipady*=8,

*row*=1,

*column*=0,

*columnspan*=4,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        self.select\_btn = tk.Button(

            root,

*text*="SELECT FILE",

*command*=self.selectfile\_callback,

*width*=42,

*bg*="#1089ff",

*fg*="#ffffff",

*bd*=2,

*relief*=tk.FLAT

        )

        self.select\_btn.grid(

*padx*=15,

*pady*=8,

*ipadx*=24,

*ipady*=6,

*row*=2,

*column*=0,

*columnspan*=4,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        self.key\_entry\_label = tk.Label(

            root,

*text*="Enter Secret Key (Remember this for Decryption)",

*bg*="#eeeeee",

*anchor*=tk.W

        )

        self.key\_entry\_label.grid(

*padx*=12,

*pady*=(8, 0),

*ipadx*=0,

*ipady*=1,

*row*=3,

*column*=0,

*columnspan*=4,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        self.key\_entry = tk.Entry(

            root,

*textvariable*=self.\_secret\_key,

*bg*="#fff",

*exportselection*=0,

*relief*=tk.FLAT

        )

        self.key\_entry.grid(

*padx*=15,

*pady*=6,

*ipadx*=8,

*ipady*=8,

*row*=4,

*column*=0,

*columnspan*=4,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        self.encrypt\_btn = tk.Button(

            root,

*text*="ENCRYPT",

*command*=self.encrypt\_callback,

*bg*="#ed3833",

*fg*="#ffffff",

*bd*=2,

*relief*=tk.FLAT

        )

        self.encrypt\_btn.grid(

*padx*=(15, 6),

*pady*=8,

*ipadx*=24,

*ipady*=6,

*row*=7,

*column*=0,

*columnspan*=2,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        self.decrypt\_btn = tk.Button(

            root,

*text*="DECRYPT",

*command*=self.decrypt\_callback,

*bg*="#00bd56",

*fg*="#ffffff",

*bd*=2,

*relief*=tk.FLAT

        )

        self.decrypt\_btn.grid(

*padx*=(6, 15),

*pady*=8,

*ipadx*=24,

*ipady*=6,

*row*=7,

*column*=2,

*columnspan*=2,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        self.reset\_btn = tk.Button(

            root,

*text*="RESET",

*command*=self.reset\_callback,

*bg*="#aaaaaa",

*fg*="#ffffff",

*bd*=2,

*relief*=tk.FLAT

        )

        self.reset\_btn.grid(

*padx*=15,

*pady*=(4, 12),

*ipadx*=24,

*ipady*=6,

*row*=8,

*column*=0,

*columnspan*=4,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        self.status\_label = tk.Label(

            root,

*textvariable*=self.\_status,

*bg*="#eeeeee",

*anchor*=tk.W,

*justify*=tk.LEFT,

*relief*=tk.FLAT,

*wraplength*=350

        )

        self.status\_label.grid(

*padx*=12,

*pady*=(0, 12),

*ipadx*=0,

*ipady*=1,

*row*=9,

*column*=0,

*columnspan*=4,

*sticky*=tk.W+tk.E+tk.N+tk.S

        )

        tk.Grid.columnconfigure(root, 0, *weight*=1)

        tk.Grid.columnconfigure(root, 1, *weight*=1)

        tk.Grid.columnconfigure(root, 2, *weight*=1)

        tk.Grid.columnconfigure(root, 3, *weight*=1)

    def selectfile\_callback(*self*):

*try*:

            name = filedialog.askopenfile()

            self.\_file\_url.set(name.name)

*# print(name.name)*

*except* Exception *as* e:

            self.\_status.set(e)

            self.status\_label.update()

    def freeze\_controls(*self*):

        self.file\_entry.configure(*state*="disabled")

        self.key\_entry.configure(*state*="disabled")

        self.select\_btn.configure(*state*="disabled")

        self.encrypt\_btn.configure(*state*="disabled")

        self.decrypt\_btn.configure(*state*="disabled")

        self.reset\_btn.configure(*text*="CANCEL", *command*=self.cancel\_callback,

*fg*="#ed3833", *bg*="#fafafa")

        self.status\_label.update()

    def unfreeze\_controls(*self*):

        self.file\_entry.configure(*state*="normal")

        self.key\_entry.configure(*state*="normal")

        self.select\_btn.configure(*state*="normal")

        self.encrypt\_btn.configure(*state*="normal")

        self.decrypt\_btn.configure(*state*="normal")

        self.reset\_btn.configure(*text*="RESET", *command*=self.reset\_callback,

*fg*="#ffffff", *bg*="#aaaaaa")

        self.status\_label.update()

    def encrypt\_callback(*self*):

        self.freeze\_controls()

*try*:

            self.\_cipher = EncryptionTool(

                self.\_file\_url.get(),

                self.\_secret\_key.get(),

                self.\_salt.get()

            )

*for* percentage *in* self.\_cipher.encrypt():

*if* self.should\_cancel:

*break*

                percentage = "{0:.2f}%".format(percentage)

                self.\_status.set(percentage)

                self.status\_label.update()

            self.\_status.set("File Encrypted!")

*if* self.should\_cancel:

                self.\_cipher.abort()

                self.\_status.set("Cancelled!")

            self.\_cipher = None

            self.should\_cancel = False

*except* Exception *as* e:

*# print(e)*

            self.\_status.set(e)

        self.unfreeze\_controls()

    def decrypt\_callback(*self*):

        self.freeze\_controls()

*try*:

            self.\_cipher = EncryptionTool(

                self.\_file\_url.get(),

                self.\_secret\_key.get(),

                self.\_salt.get()

            )

*for* percentage *in* self.\_cipher.decrypt():

*if* self.should\_cancel:

*break*

                percentage = "{0:.2f}%".format(percentage)

                self.\_status.set(percentage)

                self.status\_label.update()

            self.\_status.set("File Decrypted!")

*if* self.should\_cancel:

                self.\_cipher.abort()

                self.\_status.set("Cancelled!")

            self.\_cipher = None

            self.should\_cancel = False

*except* Exception *as* e:

*# print(e)*

            self.\_status.set(e)

        self.unfreeze\_controls()

    def reset\_callback(*self*):

        self.\_cipher = None

        self.\_file\_url.set("")

        self.\_secret\_key.set("")

        self.\_salt.set("")

        self.\_status.set("---")

    def cancel\_callback(*self*):

        self.should\_cancel = True

    def show\_help\_callback(*self*):

        messagebox.showinfo(

            "How To",

            """1. Open the App and Click SELECT FILE Button and select your file e.g. "abc.jpg".

2. Enter your Secret Key (This can be any alphanumeric letters). Remember this so you can Decrypt the file later.

3. Click ENCRYPT Button to encrypt. A new encrypted file with ".kryp" extention e.g. "abc.jpg.kryp" will be created in the same directory where the "abc.jpg" is.

4. When you want to Decrypt a file you, will select the file with the ".kryp" extention and Enter your Secret Key which you chose at the time of Encryption. Click DECRYPT Button to decrypt. The decrypted file will be of the same name as before with the suffix "\_\_dekrypted\_\_" e.g. "abc\_\_dekrypted\_\_.jpg".

5. Click RESET Button to reset the input fields and status bar.

6. You can also Click CANCEL Button during Encryption/Decryption to stop the process."""

        )

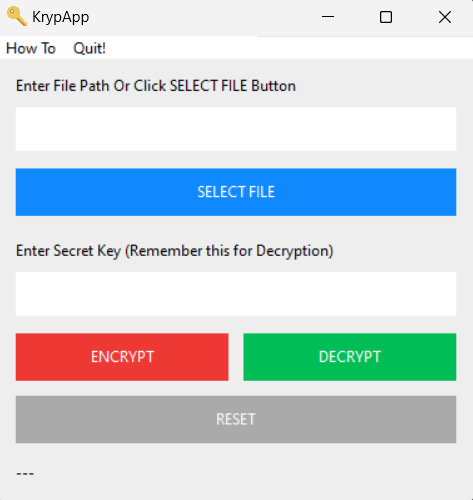
*if* \_\_name\_\_ == "\_\_main\_\_":

    ROOT = tk.Tk()

    MAIN\_WINDOW = MainWindow(ROOT)

    ROOT.mainloop()

**2.3 Outcome:**

****

**Conclusion**

From the observation it has been observed that by increasing the key length better secured system can be achieved. Longer key lengths consume more power and dissipate more heat. Basically it is a trade off between security and overheads. In order to achieve more secured system continuous efforts are required. An efficient encryption algorithm should consist of two factors – fast response and reduced complexity. A proposed direction for the future work could be to analyze the performance/security tradeoff in greater depth. For instance, an algorithm with more complex rounds and a larger number of rounds is generally considered more secure. The impact of these and other such factors on the overall performance of an algorithm needs to be measured. The work can be extended if more number of keys are used for the encryption process and then implemented on DSP

**Reference :**

* [1] Cryptography and Network Security Principles and Practices, Fourth Edition, By William Stallings.
* [2] Classification of IDEA and RSA ciphers, by M. Brahmji Rao,phd thesis,IIT Kanpur
* [3] Ritu Agarwal, Dhiraj Dafouti, Shobha Tyagi, Peformance Analysis Of Data Encryption Algorithms, O P Verma, 2011 IEEE.
* [4]<https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#:~:text=A%20random%20forest%20regressor.,accuracy%20and%20control%20over%2Dfitting>
* [5] CISSP All-in-One Certification Exam Guide200075\_ch08\_HarrisX 11/30/01 10:22 AM Page 495
* [6] National Bureau of Standards – Data Encryption Standard, FIPS Publication 46, 1977. [
* [7] [www.schneier.com/paper-twofish-fpga.pdf](http://www.schneier.com/paper-twofish-fpga.pdf)
* [8] NIST Advanced Encryption Standard (AES), Development Effort web site http://csrs.nist.gov/encryption/aes/aes-home.htm [9] C. Boyd. ―Modern Data Encryption,‖ Electronics & Communication Engineering Journal, October 1993, Vol. 5, pp 271-278.
* [10] G.Gong and A. Hasan, "An Efficient Algorithm for Exponentiation in DH Key Exchange and DSA in Cubic Extension Fields", Faculty of Mathematics, University Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp.1967 – 1969, 2010.