



# End-to-end Symmetric Encryption tool in Python

Authors: Jack Sprague, Michael Shteynberg, Ben Yaker

Northeastern University

EECE 2140. Computing Fundamentals for Engineers

Fall 2024

11/19/2024

# Abstract

In today's technological world, encryption is utilized for sending information between two users, however, existing tools are more complex or cannot be easily accessible to the average person. This project addressed the need for a simple encryption tool by using a symmetric encryption system with a GUI. This project was meant for secure messages for both the sender and receiver, in small projects or applications.

The methodology integrated Python libraries such as cryptography (Fernet, Hazmat, RSA) and Tkinter to implement the built in encryption functions and design a normal looking user interface. Symmetric encryption was focused for its simplicity and practicality for the project. The code generates a key to encrypt and decrypt text of any kind, ensuring security as long as the key remains hidden. An implementation of asymmetric encryption was also included but left out for more development and finishing touches.

Our findings include the creation of a working encryption and decryption tool that allows users to use the given key and securely encrypt and decrypt a wanted message. The GUI brings what seems to be complex code to a user, and makes it seamless to use. The limitations were discussed, such as the possible risk of the key being publicized in symmetric encryption and limited time prevented the full access of asymmetric encryption for users.

The project was built to bridge the gap between the world of coding experts, and those who prefer to use their applications. Our future work would include an updated asymmetric feature, but even without it we were still able to complete our main goal.

# Contents

1. Introduction
  - 1.1 Background
  - 1.2 Problem Statement
  - 1.3 Objectives
  - 1.4 Scope
2. Technical Approaches and Code UML
  - 2.1 Development Environment
  - 2.2 Data Collection and Preparation
  - 2.3 Implementation Details
3. Project Demonstration
  - 3.1 Screenshots and Code Snippets
4. Discussion and Future Work
5. Conclusion

# Chapter 1

## Introduction

### 1.1 Background

Encryption is an essential tool for securing sensitive information in today's digital age. While advanced encryption technologies are widely available, they often require technical expertise, making them less accessible to the average user. Our project focuses on developing a simple and easy-to-use encryption tool that can encrypt and decrypt text using symmetric encryption. By integrating a graphical user interface, we aim to make this technology more approachable for non-experts.

### 1.2 Problem Statement

Many existing encryption tools are either too complex for everyday users or lack user-friendly interfaces. This creates a barrier for individuals and small teams who need quick and reliable encryption solutions. Our project balances ease of use with security while avoiding being too complex.

### 1.3 Objectives

The main objectives of this project are:

1. To design a Python-based tool for encrypting and decrypting text using symmetric encryption.
2. To implement a user-friendly graphical interface that simplifies the process of encryption and decryption.
3. To ensure the tool maintains a basic level of security suitable for everyday use.

### 1.4 Scope

This tool is designed for educational and small-scale practical applications. It focuses on encrypting text messages rather than files or other data types. While it provides fundamental security features, it is not intended for high-stakes use cases involving advanced threats or large-scale deployment. Future improvements could include features like secure key management or support for additional encryption algorithms.

# Chapter 2

## Technical Approaches and Code UML

### 2.1 Development Background

Throughout a project a multitude of libraries and functions were used in order to correctly create a decryption and encryption code. The following libraries included cryptography (as well as .hazmat, .primitives, and .backends), fernet, a base64 bit rate, and tkinter. The specified cryptography libraries were used for hashing and padding for the symmetric encryption. In fernet, we used `fernet.generate_key`, `f.encrypt/decrypt`. Tkinter used `tk.label`, `tk.frame`, `tk.entry`, and `tk.button`.

### 2.2 Data Collection and Preparation

Our project did not include any data collection, as we did not use any data to interpret our created visualizations.

### 2.3 Implementation Details

To begin the coding process, we decided between symmetric and asymmetric encryption. After finalizing that symmetric encryption was more feasible, the asymmetric was built but not implemented in the final design. We created a code that would make a key that could be used to encrypt and decrypt a message. The next part was creating a GUI that would allow for an input of text to encrypt or decrypt. The final step was to import both the class of symmetric and functions of the GUI into a main file, to test if our work was complete. The algorithms used fernet algorithms in the symmetric encryption/decryption, as well as RSA for asymmetric encryption.

# Chapter 3

## Project Demonstration

### 3.1 Screenshots and Code Snippets

## GUI:

```
1 import tkinter as tk
2 from crypto import symCrypt
3
4 # Function to toggle between encryption and decryption frames
5 def toggle_frame():
6     if encryption_frame.winfo_ismapped(): # If the encryption frame is currently displayed
7         decryption_frame.grid(row=0, column=0, sticky='nsew') # Show the decryption frame
8         encryption_frame.grid_forget() # Hide the encryption frame
9         toggle_button.config(text="Switch to Encryption") # Update button text
10    else: # If the decryption frame is displayed
11        encryption_frame.grid(row=0, column=0, sticky='nsew') # Show the encryption frame
12        decryption_frame.grid_forget() # Hide the decryption frame
13        toggle_button.config(text="Switch to Decryption") # Update button text
14
15 sc = symCrypt()
16
17 # Function triggered when the Encrypt button is pressed
18 def encrypt_button_pressed():
19     password = recipient_public_key_entry.get() # Get the password from the input field
20     data = encryption_entry.get() # Get the text to encrypt from the input field
21     encrypted_text = sc.encrypt(data, password) # Encrypt the data using the password
22     encrypted_text_entry.delete(0, tk.END) # Clear the output field
23     encrypted_text_entry.insert(0, encrypted_text) # Display the encrypted text in the output field
24
25 # Function triggered when the Decrypt button is pressed
26 def decrypt_button_pressed():
27     password = private_key_entry.get() # Get the password from the input field
28     token = decryption_entry.get() # Get the encrypted message from the input field
29     decrypted_text = sc.decrypt(token, password) # Decrypt the message using the password
30     decrypted_text_entry.delete(0, tk.END) # Clear the output field
31     decrypted_text_entry.insert(0, decrypted_text) # Display the decrypted text in the output field
32
33 # Create the main application window
34 window = tk.Tk()
35 window.title('Encryption Tool') # Set the title of the application
36 window.geometry("500x400") # Set the dimensions of the application window
37
38 # Create separate frames for encryption and decryption
39 encryption_frame = tk.Frame(window) # Frame for encryption operations
40 decryption_frame = tk.Frame(window) # Frame for decryption operations
41
42 # Toggle button to switch between encryption and decryption modes
43 toggle_button = tk.Button(window, text="Switch to Decryption", command=toggle_frame)
44
45 toggle_button.grid(row=1, column=0, columnspan=2) # Position the button
46
47 # Configure the Encryption Frame
48 tk.Label(encryption_frame, text="Password: ").grid(row=1, column=0) # Label for password input
49 recipient_public_key_entry = tk.Entry(encryption_frame, width=30) # Entry field for password
50 recipient_public_key_entry.grid(row=1, column=1)
51
52 tk.Label(encryption_frame, text="Message to Encrypt: ").grid(row=2, column=0) # Label for message input
53 encryption_entry = tk.Entry(encryption_frame, width=30) # Entry field for message
54 encryption_entry.grid(row=2, column=1)
55
56 encrypt_button = tk.Button(encryption_frame, text="Encrypt", command=encrypt_button_pressed) # Encrypt button
57 encrypt_button.grid(row=3, column=0, columnspan=2) # Position the button
58 encryption_result_label = tk.Label(encryption_frame, text="Encrypted Text:") # Label for encrypted text
59 encryption_result_label.grid(row=4, column=0, sticky='e')
60 encrypted_text_entry = tk.Entry(encryption_frame, width=30) # Output field for encrypted text
61 encrypted_text_entry.grid(row=4, column=1, sticky='w')
62
63 # Configure the Decryption Frame
64 tk.Label(decryption_frame, text="Password: ").grid(row=0, column=0) # Label for password input
65 private_key_entry = tk.Entry(decryption_frame, width=30) # Entry field for password
66 private_key_entry.grid(row=0, column=1)
67
68 tk.Label(decryption_frame, text="Encrypted Message: ").grid(row=1, column=0) # Label for encrypted message input
69 decryption_entry = tk.Entry(decryption_frame, width=30) # Entry field for encrypted message
70 decryption_entry.grid(row=1, column=1)
71
72 decrypt_button = tk.Button(decryption_frame, text="Decrypt", command=decrypt_button_pressed) # Decrypt button
73 decrypt_button.grid(row=2, column=0, columnspan=2) # Position the button
74 decryption_result_label = tk.Label(decryption_frame, text="Decrypted Text: ") # Label for decrypted text
75 decryption_result_label.grid(row=4, column=0, sticky='e')
76 decrypted_text_entry = tk.Entry(decryption_frame, width=30) # Output field for decrypted text
77 decrypted_text_entry.grid(row=4, column=1, sticky='w')
78
79 # Initially display the encryption frame
80 encryption_frame.grid(row=0, column=0, sticky='nsew')
```

## Symmetric:

```
import base64
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC

salt = b"thesalt"

class symCrypt:

    def encrypt(self, data, password):
        key = deriveKey(password, 32, salt)
        token = sym_encrypt(data, key)
        return token

    def decrypt(self, token, password):
        key = deriveKey(password, 32, salt)
        data = sym_decrypt(token, key)
        return data

#returns a new generated a 32-byte fernet key for symmetric encryption
def generateSymKey():
    key = Fernet.generate_key()
    return key.decode('ascii')

#returns encrypted data using a key with the fernet symmetric encryption algorithm
def sym_encrypt(data, key):
    data = data.encode('ascii')
    key = key.encode('ascii')

    f = Fernet(key)
    return f.encrypt(data).decode('ascii')

#returns decrypted data using a key with the fernet symmetric encryption algorithm
def sym_decrypt(token, key):
    token = token.encode('ascii')
    key = key.encode('ascii')

    f = Fernet(key)
    return f.decrypt(token).decode('ascii')

#derives a key from a password string
def deriveKey(password, length, salt):
    kdf = PBKDF2HMAC(
```

```
        algorithm=hashes.SHA256(),
        length=length,
        salt=salt,
        iterations=480000,
    )

    #key = kdf.derive(password.encode('ascii'))
    key = base64.urlsafe_b64encode(kdf.derive(password.encode('ascii')))

    return key.decode('ascii')
```



## Asymmetric:

```
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.backends import default_backend

class AsymKey:
    def __init__(self, keysize=1024):
        # Initialize the class, creating a bit size (keysize), and using the rsa (a public key encryption method)
        # built (variable) private_key: RSAPrivateKey, which can have different bit size
        # or y used.
        self.private_key = rsa.generate_private_key(
            public_exponent=65537,
            key_size=keysize,
            backend=default_backend()
        )
        self.public_key = self.private_key.public_key()
        # The public key is generated using the private key, that way they are linked, by simply
        # implementing the .public_key function thats built in

    def serialize_private(self, password):
        # This function uses serialization, which allows us to store the keys into files
        if password is None:
            # Leaving the password as nothing will create key
            encryption_algorithm = serialization.NoEncryption()
        else:
            # Or we can include a password which will also use the password to create a different key
            encryption_algorithm = serialization.BestAvailableEncryption(password.encode())
        # The if statements create an algorithm based of the password on the instance

        # private_bytes lets us pick the type of serialization and format we want to use
        return self.private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=encryption_algorithm
        )

    def serialize_public(self):
        # The public key serialization is done the same, however it doesnt need the password, since we base the public key
        # off the private key in the beginning
        return self.public_key.public_bytes(
```

```
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo
        )

    def key_to_file(self, datakey, file_path):
        with open(file_path, 'wb') as file:
            file.write(datakey)
        # After creating and serializing the keys, write them into pem files, and create a fuction that will be
        # used in the following

    def save_private(self, file_path, password):
        # To save the key, we use the serialize function, and the key to file function
        datakey=self.serialize_private(password=password)
        self.key_to_file(datakey, file_path)

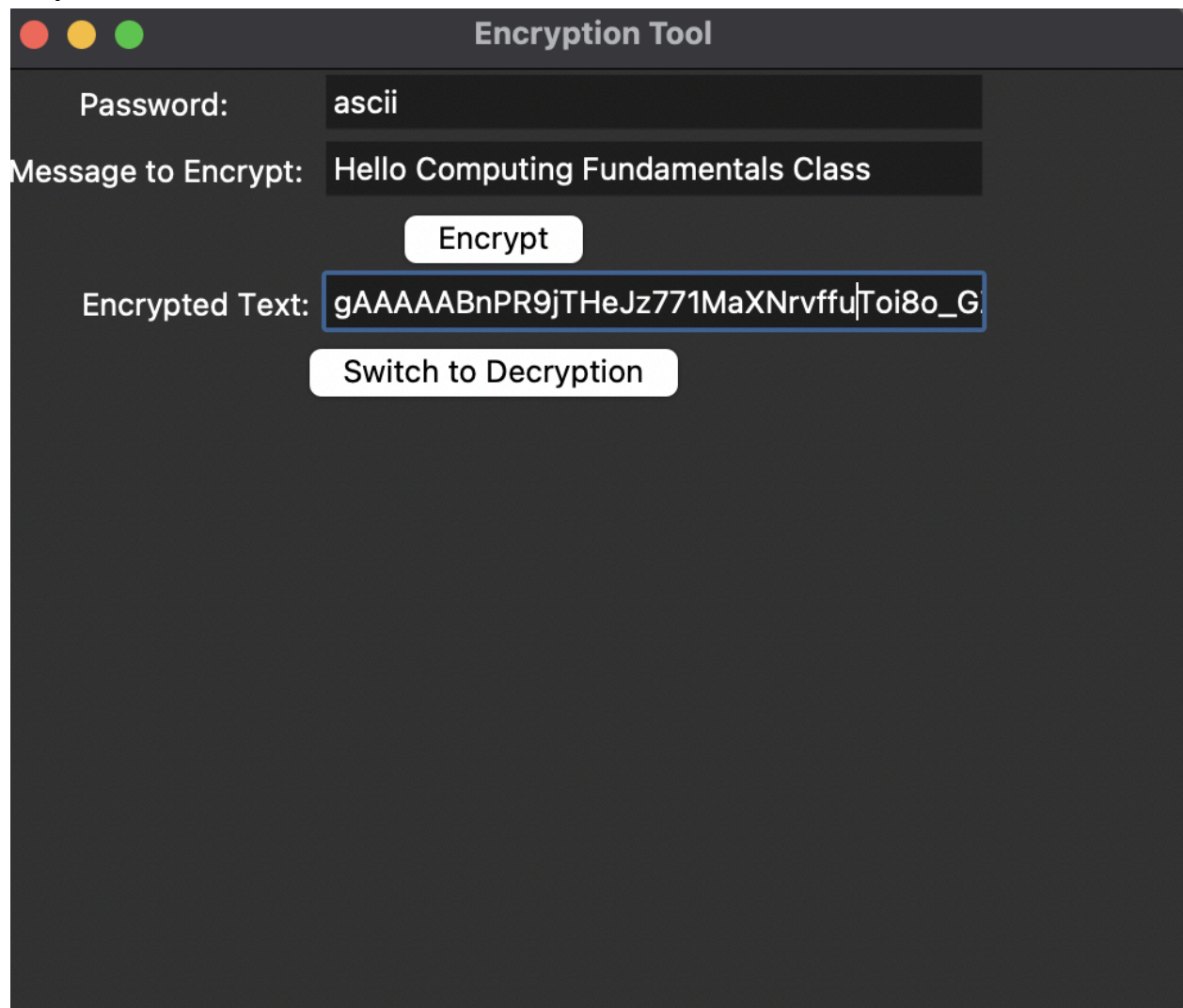
    def save_public(self, file_path):
        # To save the public key we do the same as the private, however we dont need the password
        datakey=self.serialize_public()
        self.key_to_file(datakey, file_path)

keyJack = AsymKey()
keyJack.save_private('private_key_jack.pem', password="Jack")
keyJack.save_public("public_key_jack.pem")

keyMicheal = AsymKey()
keyMicheal.save_private('private_key_micheal.pem', password="Micheal")
keyMicheal.save_public("public_key_micheal.pem")

keyBen = AsymKey()
keyBen.save_private('private_key_ben.pem', password="Ben")
keyBen.save_public("public_key_ben.pem")
```

Output:



The image shows a macOS-style application window titled "Encryption Tool". It has a dark gray background and standard macOS window controls (red, yellow, green buttons) in the top-left corner. The interface contains the following elements:

- Password:** A text input field containing the word "ascii".
- Message to Encrypt:** A text input field containing the text "Hello Computing Fundamentals Class".
- Encrypt:** A white button with rounded corners and the text "Encrypt".
- Encrypted Text:** A text input field containing the string "gAAAAABnPR9jTHeJz771MaXNrvffu|Toi8o\_G". This field is highlighted with a blue border.
- Switch to Decryption:** A white button with rounded corners and the text "Switch to Decryption".

Encryption Tool

Password:

ascii

Encrypted Message:

3x29R6lp9qKLclzK-3ekhX3buehPFjtB1Afxn

Decrypt

Decrypted Text:

Hello Computing Fundamentals Class

Switch to Encryption

# Chapter 4

## Discussion and Future Work

The final results of our project is that our encryption and decryption software works! We are able to add text to the interface and secretly send messages between people, with the only thing needed to make it work being the symmetric key. The biggest implication we had was our unfamiliarity with the libraries needed to create this encryption tool. We were forced to learn what specific functions did, how we could implement them, and overall to correctly design an interface with encryption and decryption. Limitations we found as created this tool, was being able to use both symmetric and asymmetric within our project, in the given amount of time. On top of that, if the symmetric password were to be leaked, our secretive messages would no longer be so secret. This was the reason we wanted to include the asymmetric feature to have a secure version for our tool. As a result, the asymmetric was made, but planned as a future work/implementation.

# Chapter 5

## Conclusion

Our project successfully created an end-to-end symmetric encryption tool using Python, with the capability of encrypting and decrypting text via a GUI. The primary goal of creating a basic, easy to use encryption tool was achieved by implementing symmetric encryption with the Fernet module built in python's cryptography library and designing the interface with Tkinter.

Key findings:

- The symmetric encryption is functional, allowing users to encrypt and decrypt text with the click of a button.
- The tool is intuitive enough for non coding users, making encryption of small applications fast and easy.
- The beginning step toward incorporating asymmetric encryption was taken, but the inclusion of the encryption was saved for the future.

The prioritization of this project stands in its ability to take the gap away between what seems to be complex encryption and user-friendly design. It depicts how cryptography can be applied to create accessible solutions for communication security. While limiting factors, such as symmetric key security, were clearly outlined, the project created a base for future work, including asymmetric encryption for a more secure method.

Overall, the project achieved its initial purpose of enabling users encryption while adding a quick and easy manner to do so.

# Bibliography

Advocate, Dan AriasStaff Developer, and Dan AriasStaff Developer AdvocateHowdy! ? I do technology research at Auth0 with a focus on security and identity and develop apps to showcase the advantages or pitfalls of such technology. I also contribute to the development of our SDKs.

“Adding Salt to Hashing: A Better Way to Store Passwords.” *Auth0*, 25 Feb. 2021, [auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/](https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/).

“Page.” *GuiProgramming - Python Wiki*, [wiki.python.org/moin/GuiProgramming](https://wiki.python.org/moin/GuiProgramming). Accessed 19 Nov. 2024.

“Python Tkinter.” *GeeksforGeeks*, GeeksforGeeks, 19 June 2024, [www.geeksforgeeks.org/python-gui-tkinter/](https://www.geeksforgeeks.org/python-gui-tkinter/).

“Welcome to Pyca/Cryptography.” *Welcome to Pyca/Cryptography - Cryptography 44.0.0.Dev1 Documentation*, [cryptography.io/en/latest/](https://cryptography.io/en/latest/). Accessed 19 Nov. 2024.