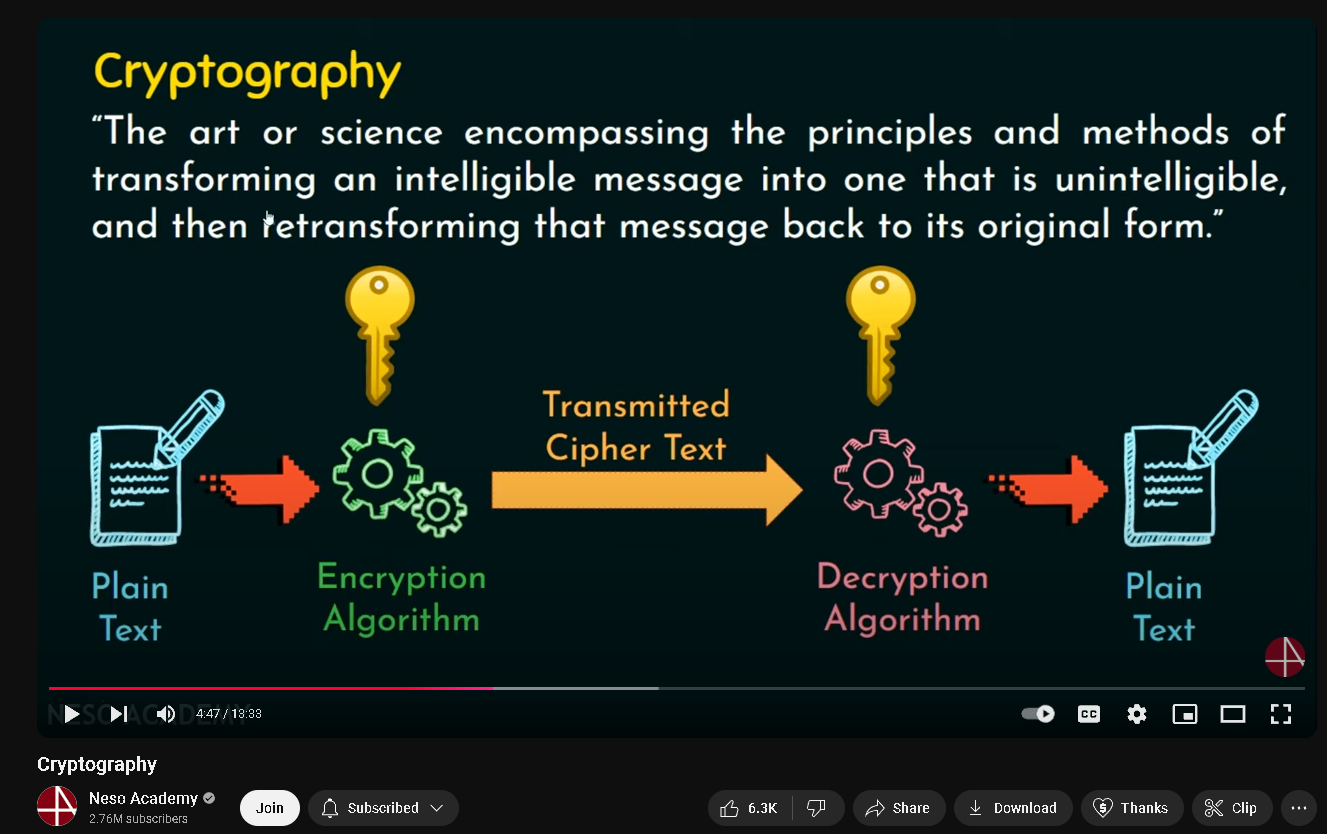
Analyze existing cryptographic algorithms or protocols using automata theory. Develop new cryptographic techniques based on automata theory.  
  
Cryptography:



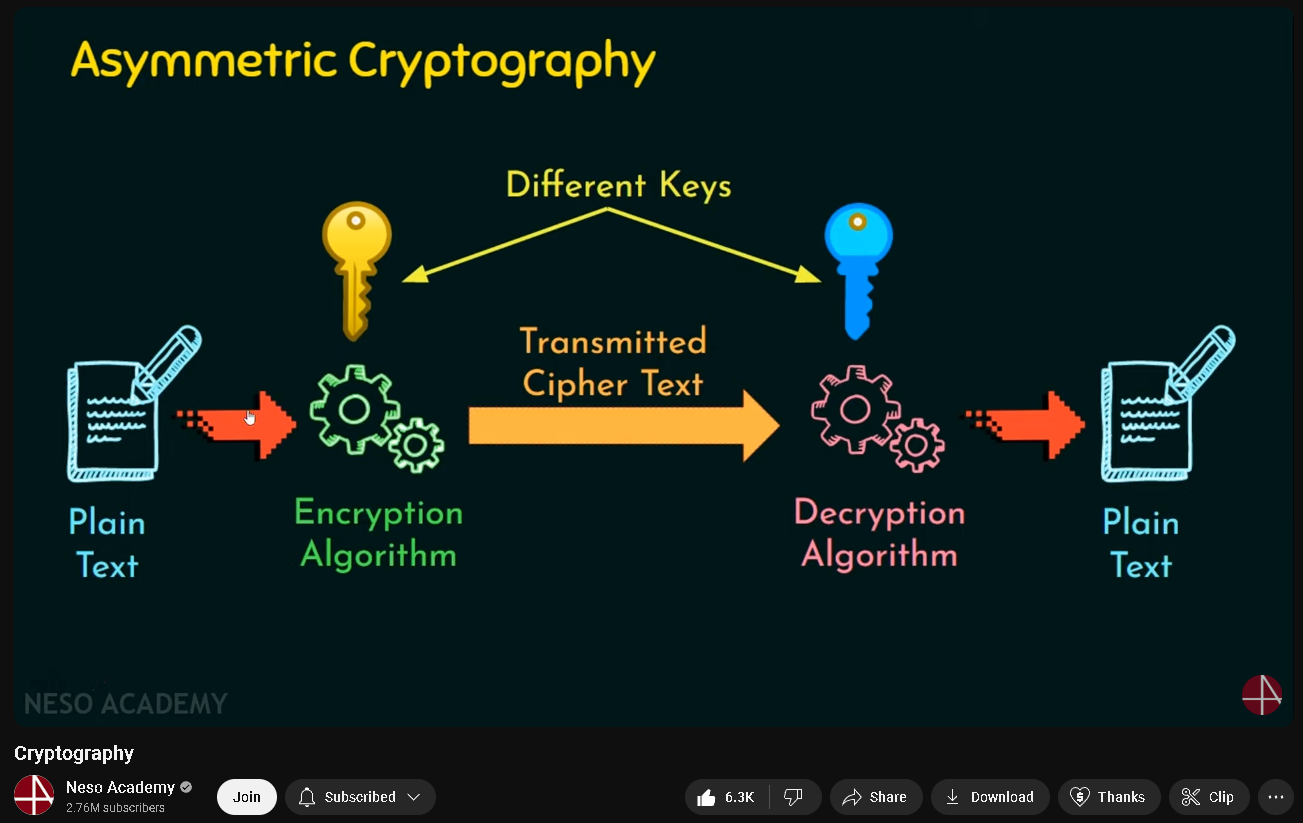
A screenshot of a video

Description automatically generated

A screenshot of a computer

Description automatically generated

Private key ^^^^



Public key ^^^  
  
  
RSA:  
A person writing on a whiteboard

Description automatically generated

“Analyze existing cryptographic algorithms or protocols using automata theory.”

“Develop new cryptographic techniques based on automata theory.”

Bandiala, Xyrujon Alfred M. Automata Theory and Formal Languages

CS-3C Final Project (PIT)

**Analysis with automata theory.**

We’ll be analyzing both Symmetric Cryptography and Asymmetric Cryptography (RSA) to understanding cryptography through the lens of automata theory.

I’ll be using concepts that I am most knowledgeable on to see how it can really relate to the cryptographic algorithms or protocols. I’ll be doing it with Pushdown Automata.

The 2nd cryptographic technique we’ll be analyzing will be one of the common and simple encryption techniques which is the Caesar-cipher with the use of FA or Finite Automata which is another concept that I am very confident on to be using.

Certainly! Let me explain the selected code, which implements the Caesar cipher encryption:

```python

def caesar\_encrypt(text, shift):

"""Encrypts text using Caesar cipher."""

encrypted = ""

for char in text:

if char.isalpha():

shift\_base = 65 if char.isupper() else 97

encrypted += chr((ord(char) - shift\_base + shift) % 26 + shift\_base)

else:

encrypted += char

return encrypted

```

This function, `caesar\_encrypt`, implements the Caesar cipher encryption algorithm. Here's a breakdown of how it works:

1. The function takes two parameters: `text` (the message to be encrypted) and `shift` (the number of positions to shift each letter).

2. It initializes an empty string `encrypted` to store the encrypted message.

3. The function iterates through each character in the input `text`:

a. If the character is alphabetic (checked using `char.isalpha()`), it applies the Caesar cipher shift:

- It determines the `shift\_base` as 65 for uppercase letters (ASCII 'A') or 97 for lowercase letters (ASCII 'a').

- It then applies the formula `(ord(char) - shift\_base + shift) % 26 + shift\_base` to shift the character:

\* `ord(char) - shift\_base` converts the letter to a 0-25 range.

\* `+ shift` applies the desired shift.

\* `% 26` ensures the result wraps around the alphabet.

\* `+ shift\_base` converts the result back to the ASCII range.

- The shifted character is then converted back to a character using `chr()` and added to the `encrypted` string.

b. If the character is not alphabetic, it's added to the `encrypted` string without any change.

4. Finally, the function returns the fully encrypted string.

This implementation preserves the case of the original text and only encrypts alphabetic characters, leaving numbers, punctuation, and other characters unchanged.

(OG\_encryption\_Triple)

The selected code defines a function called `triple\_encrypt` within the `og encrypt.py` file. This function applies three layers of encryption to a given plaintext.

Here's a breakdown of the selected code:

1. The function `triple\_encrypt` takes two parameters: `plaintext` (the text to be encrypted) and `caesar\_shift` (the shift value for the Caesar cipher; default is 3).

2. Inside the function, three layers of encryption are applied:

- The first layer uses the `caesar\_encrypt` function to encrypt the plaintext using a Caesar cipher with the specified shift value. The result is stored in the variable `layer1`.

- The second layer uses the `transposition\_encrypt` function to reverse the order of characters in `layer1`. The result is stored in the variable `layer2`.

- The third layer uses the `replace\_vowels\_encrypt` function to replace vowels in `layer2` with assigned numbers and then applies a Caesar cipher with a shift value of 2. The result is stored in the variable `layer3`.

3. Finally, the function returns `layer3`, which contains the encrypted text after applying all three layers of encryption.

This function is used to encrypt the input message and save it to a file. The main function calls `encrypt\_to\_file` with the user-provided message and the desired file name. The encrypted text is then saved to the specified file.

Certainly! I'll explain the `triple\_encrypt` function in terms of automata theory. In this context, we can view the encryption process as a series of finite state transducers (FSTs) that transform the input string.

Let's break down the `triple\_encrypt` function into three separate FSTs, each representing a layer of encryption:

1. Caesar Cipher FST (M1):

- States: Q = {q0, q1, ..., q25} (26 states, one for each possible shift)

- Input alphabet: Σ = {all printable ASCII characters}

- Output alphabet: Γ = {all printable ASCII characters}

- Transition function: δ(qi, c) = qj, where j = (i + caesar\_shift) mod 26

- Output function: λ(qi, c) = c', where c' is the character shifted by caesar\_shift

2. Transposition (Reversal) FST (M2):

- States: Q = {q0, q1}

- Input alphabet: Σ = {all printable ASCII characters}

- Output alphabet: Γ = {all printable ASCII characters}

- Transition function: δ(q0, c) = q1 (read entire input)

- Output function: λ(q1, ε) = reverse(input\_string)

3. Replace Vowels and Caesar Cipher FST (M3):

- States: Q = {q0, q1, ..., q25} (26 states for Caesar shift)

- Input alphabet: Σ = {all printable ASCII characters}

- Output alphabet: Γ = {all printable ASCII characters, 0, 1, 2, 3, 4}

- Transition function: δ(qi, c) = qj, where j = (i + 2) mod 26

- Output function: λ(qi, c) =

- '0' if c in {'a', 'A'}

- '1' if c in {'e', 'E'}

- '2' if c in {'i', 'I'}

- '3' if c in {'o', 'O'}

- '4' if c in {'u', 'U'}

- c' otherwise, where c' is c shifted by 2 (Caesar cipher)

The `triple\_encrypt` function can be seen as a composition of these three FSTs:

M = M3 ∘ M2 ∘ M1

Where ∘ denotes the composition operation of FSTs.

The overall process can be described as:

1. The input string (plaintext) is fed into M1 (Caesar Cipher FST).

2. The output of M1 becomes the input for M2 (Transposition FST).

3. The output of M2 becomes the input for M3 (Replace Vowels and Caesar Cipher FST).

4. The final output of M3 is the encrypted text.

In terms of language theory, if we denote the language of all possible input strings as L, then the `triple\_encrypt` function maps L to L', where L' is the language of all possible encrypted strings produced by this composite FST.

This automata-theoretic view helps us understand the encryption process as a series of well-defined state transitions and transformations on the input string, providing a formal framework for analyzing the encryption algorithm.