Embarking on a project is like embarking on an exciting journey! Who says you have to be a master of the topic from the start? In fact, the project itself becomes your teacher, guiding you towards mastery as you progress.

Welcome to our Python-powered Message Encryption and Decryption adventure! We're starting from scratch, delving into every nook and cranny to truly grasp the intricacies of this fascinating world.

Let's dive into the first chapter: Encryption and Decryption. Picture this: you're standing amidst a lively group, but you desperately need to convey a secret message to your best friend. How can you do it without prying eyes catching on? Fear not! Just like in those secret agent movies, we'll employ clever sign language or a mysterious code that only the two of you understand. This, my friend, is the essence of encryption in the digital realm. It's like having your very own secret language, accessible only to those with the magical key.

Now, imagine your best friend decrypting your hidden message with a grin of understanding. That, my friend, is the art of decryption - transforming that encrypted puzzle back into its original, delightful form.

In this magical realm of encryption, we call the original message the "plaintext" and its encrypted counterpart the "ciphertext." Our mission is to take you on a thrilling journey through the land of cryptography, where we'll create a system that ensures your messages remain hidden from prying eyes.

So, fasten your seatbelt, put on your detective hat, and let's unravel the mysteries of encryption and decryption together! Our adventure awaits, where secrets are locked away and only the right keys can unlock them. Get ready for a thrilling ride!

The chain of the message is like:

Sender: Plaintext -> Encryption: Ciphertext -> Decryption: Plain Text -> Receiver

What is the concept of key?

In encryption and decryption, keys are used to transform plain text into cipher text and vice versa. Keys are essentially a set of rules or parameters that are used to modify the data in a specific way so that it becomes unintelligible to unauthorized individuals.

Two types of keys:

1. **Symmetric:** In symmetric encryption, the same key is used for both encryption and decryption. This means that the sender and the receiver of the message must share the same key. The key is a secret piece of information that is known only to the sender and the receiver.

2. **Asymmetric:** In asymmetric encryption, two different keys (a public key and a private key) are used. The public key is shared with anyone who needs to send an encrypted message to the receiver, while the private key is kept secret by the receiver.

In both cases, the keys work by applying a specific algorithm to the message, transforming it into cipher text that is unintelligible to unauthorized individuals. The reverse algorithm is then used to decrypt the message, converting it back into plain text. By using keys to encrypt and decrypt messages, encryption provides a secure means of communication, ensuring that the data remains confidential.

Symmetric Encryption:

In symmetric encryption, the same key is used for both encryption and decryption. This means that the sender and the receiver of the message must share the same key. Popular symmetric encryption algorithms include **Advanced Encryption Standard (AES) and Blowfish.**

The process of symmetric encryption involves converting the plaintext into cipher text by applying a mathematical function, which uses a secret key as input. The recipient uses the same secret key to reverse the process, decrypting the cipher text and converting it back into plaintext. Because the same key is used for both encryption and decryption, symmetric encryption is generally faster and more efficient than asymmetric encryption.

Asymmetric Encryption:

In asymmetric encryption, two different keys (a public key and a private key) are used. The public key is shared with anyone who needs to send an encrypted message to the receiver, while the private key is kept secret by the receiver.

Popular asymmetric encryption algorithms include **Rivest–Shamir–Adleman (RSA) and Elliptic Curve Cryptography (ECC).**

The process of asymmetric encryption involves the sender using the receiver's public key to encrypt the message. Once the message is encrypted, only the receiver can decrypt it using their private key. Because two different keys are used, asymmetric encryption is generally slower and less efficient than symmetric encryption, but it provides greater security since the private key is kept secret.

In summary, symmetric encryption is faster and more efficient but requires both parties to have the same secret key, while asymmetric encryption is slower but more secure since it uses two different keys.

AES (Advanced Encryption Standard)

AES is a widely-used encryption algorithm that is known for its security and efficiency. It uses a block cipher to encrypt data in fixed-size blocks, and the key size can vary between 128, 192, and 256 bits.A block cipher is a method of encrypting data in blocks to produce ciphertext using a cryptographic key and algorithm .

In the project, we used the PyCrypto library in Python to implement AES encryption. Specifically, we used the AES class from the Crypto.Cipher module to create an AES object, and we used the encrypt() and decrypt() methods to encrypt and decrypt the data.

Here is a brief summary of how the AES algorithm works:

1. The plain text is broken up into fixed-size blocks of 128 bits each.
2. The AES algorithm applies a series of transformations to the data, including substitution, permutation, and mixing of the bits.
3. The key is used to modify the data in a specific way so that it becomes unintelligible to unauthorized individuals.
4. The transformed data, or cipher text, is sent to the recipient.
5. The recipient uses the same key to reverse the transformations and recover the original data.

Overall, AES is a widely-used and secure encryption algorithm that can be used to protect data in various applications.

Blowfish

Blowfish is another symmetric encryption algorithm that is widely-used and known for its security. It was developed by Bruce Schneier in 1993 and is designed to be fast, efficient, and easy to implement.

Blowfish is a block cipher that operates on blocks of 64 bits at a time, and it can use variable key lengths up to a maximum of 448 bits. The algorithm consists of a series of substitution and permutation operations that are repeated a number of times, based on the key size and the number of rounds specified.

Here is a brief summary of how the Blowfish algorithm works:

1. The plain text is broken up into fixed-size blocks of 64 bits each.
2. The key is expanded into a series of subkeys using a key schedule algorithm.
3. The subkeys are used to modify the data in a specific way, using a series of substitution and permutation operations.
4. The transformed data, or cipher text, is sent to the recipient.
5. The recipient uses the same key and algorithm to reverse the transformations and recover the original data.
6. One advantage of Blowfish is its flexibility in terms of key size, which allows for stronger encryption and improved security. Additionally, it is relatively easy to implement and can be used in a variety of applications.

Overall, Blowfish is a popular and secure encryption algorithm that can be used to protect sensitive data in various applications.

Difference between AES and Blowfish

AES and Blowfish are both symmetric encryption algorithms that use block ciphers to encrypt data. They are both widely-used and known for their security and efficiency. However, there are some key differences between the two algorithms that are worth noting.

Key size:

One major difference between AES and Blowfish is their key size. AES supports key sizes of 128, 192, or 256 bits, while Blowfish supports key sizes from 32 to 448 bits, in increments of 8 bits. This means that Blowfish supports a wider range of key sizes than AES, which can provide additional flexibility and security.

Block size:

Another difference between AES and Blowfish is their block size. AES has a fixed block size of 128 bits, while Blowfish has a variable block size of 64 bits, which can be increased up to 448 bits. This means that AES can process larger blocks of data at once, while Blowfish is more flexible in terms of block size.

Speed:

Blowfish is generally considered to be faster than AES in software implementations, especially for smaller block sizes and key sizes. However, AES is often faster than Blowfish for larger block sizes and key sizes.

Security:

Both AES and Blowfish are considered to be secure encryption algorithms, but AES is generally considered to be more secure, particularly against attacks such as brute force attacks. This is because AES has a more complex structure and more rounds of encryption than Blowfish.

Ease of implementation:

Blowfish is generally considered to be easier to implement than AES, due to its simpler structure and smaller number of rounds. This can make it a good choice for applications where simplicity and ease of implementation are important factors.

Overall, both AES and Blowfish are strong encryption algorithms that can provide good security for sensitive data. The choice between the two algorithms may depend on factors such as key size, block size, speed, security, and ease of implementation.

RSA

RSA, which stands for Rivest-Shamir-Adleman, is a widely-used asymmetric encryption algorithm that is based on the mathematical problem of factoring large numbers. The algorithm works as follows:

1. **Key generation:** The first step in using RSA is to generate a pair of keys - a public key and a private key - using a key generation algorithm. The public key can be shared with anyone, while the private key is kept secret.
2. **Encryption:** To encrypt data using RSA, the sender uses the recipient's public key to encrypt the data. The sender first breaks the message into blocks of fixed size and then applies the public key to each block using a mathematical formula. The resulting encrypted blocks, or cipher text, are sent to the recipient.
3. **Decryption:** To decrypt the cipher text, the recipient uses their private key, which is the only key that can decrypt the data. The recipient applies their private key to each block of cipher text using a mathematical formula, which recovers the original message.

One key advantage of RSA is its security, which is based on the difficulty of factoring large numbers. It is a very secure encryption algorithm, especially when used with key sizes of 2048 bits or greater. However, RSA can be slow for certain applications, especially for large data sets.

Overall, RSA is a widely-used asymmetric encryption algorithm that is known for its security and versatility, and is commonly used in various applications such as secure communication, digital signatures, and more.

ECC

ECC, or Elliptic Curve Cryptography, is another widely used asymmetric encryption algorithm. It is based on the mathematical properties of elliptic curves and provides strong security with relatively small key sizes.

The key generation process in ECC is like that of RSA, but instead of using large prime numbers, it uses points on an elliptic curve. The algorithm works as follows:

1. **Key generation:** The first step in using ECC is to generate a pair of keys - a public key and a private key. The key generation algorithm starts by selecting an elliptic curve and a point on that curve, called the base point. The parameters of the curve are typically selected to provide a balance between security and efficiency. The private key is then a random number between 1 and the order of the base point, while the public key is the result of multiplying the base point by the private key using a point multiplication algorithm. The resulting public key is a point on the curve.

2. **Encryption:** To encrypt data using ECC, the sender first converts the message into a point on the elliptic curve using a mathematical formula. The formula is M * G, where M is the message, G is the base point, and the multiplication is performed using point multiplication algorithm. The resulting point is then multiplied by the recipient's public key to produce the cipher text. The recipient's public key is also a point on the curve, so the multiplication is performed using the point multiplication algorithm.

3. **Decryption:** To decrypt the cipher text, the recipient multiplies their private key by the cipher text point using the point multiplication algorithm. The resulting point is then multiplied by the base point to recover the original message. This is because the private key is the inverse of the public key, so multiplying by the private key "cancels out" the multiplication by the public key.

ECC has several advantages over RSA, including smaller key sizes, faster computations, and resistance to certain types of attacks. However, it is also more complex to implement and requires careful selection of curve parameters to ensure security. ECC is widely used in applications such as secure communication, digital signatures, and mobile device security.

In summary, ECC is an asymmetric encryption algorithm that is based on the properties of elliptic curves. It provides strong security with relatively small key sizes and is widely used in secure communication and other applications. The key generation process involves selecting an elliptic curve and a base point, and then generating public and private keys by performing point multiplication.

We will be moving forward with Blowfish

Let's start making project.

The step1 of the problem is to choose an encryption algorith, which we have already done. We will be using Blowfish.

Step 2: Open a web browser: Launch your preferred web browser (e.g., Google Chrome, Mozilla Firefox).

Step 3: Go to the PyPI page: Visit the PyPI (Python Package Index) page for pycryptodome by going to the following URL: https://pypi.org/project/pycryptodome/

Step 4: Download the latest version: On the PyPI page, locate the "Download files" section. Look for the .tar.gz or .zip file associated with the latest version of pycryptodome. Click on the file to download it to your computer. Make sure to select the appropriate version compatible with your Python installation (e.g., Python 3.7, 3.8, etc.).

Step 5: Extract the downloaded file: Once the download is complete, navigate to the downloaded file on your computer. Right-click on the file and select an option like "Extract All" or "Extract Here" to extract its contents. This will create a new folder with the extracted files.

Step 6: Open the Command Prompt: Press the Windows key on your keyboard and type "cmd" (without quotes) in the search bar. Press Enter to open the Command Prompt.

Step 7: Navigate to the extracted folder: In the Command Prompt, use the cd command to navigate to the folder where you extracted the pycryptodome files. For example, if you extracted the files to the C:\Downloads\pycryptodome folder, you would use the following command:

```
cd C:\Downloads\pycryptodome
```

Step 8: Install pycryptodome: With the Command Prompt still open and the correct folder navigated to, run the following command to install pycryptodome:

```
pip install pycroptodome
```

Step 9: verify the installation: Once the installation completes, you can verify it by importing Crypto in a Python script or the Python interactive shell. Open a Python interpreter by typing python in the Command Prompt and pressing Enter. Then, type the following command:

```
from Crypto.Cipher import Blowfish
```

Note: If even after following all the steps, pycryptodome is not working and there is any difficulty in downloading or importing the package. Try extrcating the zipped file in C drive then copy path in command prompt. Its easy to work on that this way. And also keep a copy of the files in your project folder that will ensure the successful importing of the package and the code will run smoothly.

Now here is the code

```
from Crypto.Cipher import Blowfish
from Crypto.Util.Padding import pad, unpad

def encrypt_message(message, key):
    cipher = Blowfish.new(key.encode(), Blowfish.MODE_ECB)
    padded_message = pad(message.encode(), Blowfish.block_size)
    ciphertext = cipher.encrypt(padded_message)
    return ciphertext.hex()

def decrypt_message(ciphertext, key):
    cipher = Blowfish.new(key.encode(), Blowfish.MODE_ECB)
    decrypted_message = cipher.decrypt(bytes.fromhex(ciphertext))
    message = unpad(decrypted_message, Blowfish.block_size).decode()
    return message

# Example usage:
message = input("Enter a message to encrypt: ")
key = input("Enter an encryption key: ")

encrypted_message = encrypt_message(message, key)
print("Encrypted message:", encrypted_message)

decrypted_message = decrypt_message(encrypted_message, key)
print("Decrypted message:", decrypted_message)
```

Now let’s understand the code:

1. **from Crypto.Cipher import Blowfish**

from Crypto.Util.Padding import pad, unpad

This line imports the necessary modules from the Crypto package. Specifically, it imports the Blowfish cipher implementation from Crypto.Cipher and the padding functions pad and unpad from Crypto.Util.Padding. These modules provide the tools needed for encryption and decryption operations.

2. **def encrypt_message(message, key):**

```
        cipher = Blowfish.new(key.encode(), Blowfish.MODE_ECB)
        padded_message = pad(message.encode(), Blowfish.block_size)
         ciphertext = cipher.encrypt(padded_message)
          return ciphertext.hex()
```

This defines the encrypt_message function, which takes a message and a key as input parameters.

cipher = Blowfish.new(key.encode(), Blowfish.MODE_ECB) initializes the Blowfish cipher with the provided key. The key is encoded into bytes, and the encryption mode is set to ECB (Electronic Codebook) mode.

padded_message = pad(message.encode(), Blowfish.block_size) pads the message with additional bytes to ensure its length is a multiple of the Blowfish block size.

ciphertext = cipher.encrypt(padded_message) encrypts the padded message using the Blowfish cipher.

return ciphertext.hex() returns the hexadecimal representation of the ciphertext, making it easier to display and store.

3. **def decrypt_message(ciphertext, key):**

```
     cipher = Blowfish.new(key.encode(), Blowfish.MODE_ECB)
    decrypted_message = cipher.decrypt(bytes.fromhex(ciphertext))
   message = unpad(decrypted_message, Blowfish.block_size).decode()
  return message
```

This defines the decrypt_message function, which takes a ciphertext and a key as input parameters.

cipher = Blowfish.new(key.encode(), Blowfish.MODE_ECB) initializes the Blowfish cipher with the provided key, following the same process as in the encryption function.

decrypted_message = cipher.decrypt(bytes.fromhex(ciphertext)) decrypts the ciphertext by passing the ciphertext as bytes, obtained by converting the hexadecimal representation back to bytes.

message = unpad(decrypted_message, Blowfish.block_size).decode() removes any padding from the decrypted message and decodes it from bytes to a string.

return message returns the decrypted message.

4. **message = input("Enter a message to encrypt: ")**

   key = input("Enter an encryption key: ")

encrypted_message = encrypt_message(message, key)

print("Encrypted message:", encrypted_message)

decrypted_message = decrypt_message(encrypted_message, key)

print("Decrypted message:", decrypted_message)

These lines prompt the user to enter a message to encrypt and an encryption key.

encrypted_message = encrypt_message(message, key) calls the encrypt_message function, passing the message and key as arguments, and assigns the result to encrypted_message.

print("Encrypted message:", encrypted_message) displays the encrypted message to the user.

decrypted_message = decrypt_message(encrypted_message, key) calls the decrypt_message function, passing the encrypted message and key as arguments, and assigns the result to decrypted_message.

print("Decrypted message:", decrypted_message) displays the decrypted message to the user.