Project - Data Encryption Implementation

1. Research and Selection of Encryption Algorithms

For this task, we selected the AES-256-CBC algorithm, a symmetric key encryption algorithm known for its security and performance.

More about AES-256-CBC Algorithm

AES (Advanced Encryption Standard) is a symmetric encryption algorithm widely used across the globe to secure data. The **AES-256-CBC** variant stands for Advanced Encryption Standard with a 256-bit key and Cipher Block Chaining mode. It was chosen for the following reasons:

1. **Strong Security**: AES-256 uses a 256-bit key, making it highly secure against brute force attacks. Given the current computational capabilities, breaking AES-256 encryption is practically infeasible.

2. **Performance**: Despite its high security, AES-256-CBC offers excellent performance, making it suitable for a wide range of applications, from securing data at rest to data in transit.

3. **Widespread Adoption**: AES is widely adopted and trusted by many organizations and industries, including the US government, which has approved AES for encrypting classified information.

4. **CBC Mode**: Cipher Block Chaining (CBC) mode adds an additional layer of security by ensuring that identical plaintext blocks produce different ciphertext blocks, thereby preventing pattern analysis. In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This ensures that even if the same plaintext is encrypted multiple times, the resulting ciphertext will be different each time.

Other Encryption Methods

RSA Encryption

RSA (Rivest-Shamir-Adleman) is an asymmetric encryption algorithm that uses a pair of keys: a public key for encryption and a private key for decryption.

Strengths:- Strong security: RSA with a sufficiently long key (2048 bits or more) is considered highly secure.- Asymmetric nature: Allows for secure key exchange without the need to share a secret key.- Widely used in digital signatures and SSL/TLS protocols.

Weaknesses:- Performance: RSA is significantly slower than symmetric encryption algorithms like AES.- Key length: Requires longer keys to maintain security, which increases computational overhead.

DES Encryption

DES (Data Encryption Standard) is a symmetric key algorithm that was widely used in the past.

Strengths:- Simplicity: DES is simple to implement and was widely adopted for many years.

Weaknesses:- Security: DES uses a 56-bit key, which is no longer considered secure due to advances in computational power that can break it through brute force attacks.- Superseded: DES has been largely replaced by more secure algorithms like AES.

Triple DES Encryption

Triple DES (3DES) is an enhancement of DES that applies the DES algorithm three times to each data block.

Strengths:- Improved security: 3DES is more secure than DES due to its longer effective key length.

Weaknesses:- Performance: 3DES is slower than DES and other modern algorithms like AES.- Key length: Despite improved security, 3DES is still vulnerable to certain attacks and is being phased out in favor of AES.

Blowfish Encryption

Blowfish is a symmetric key block cipher known for its speed and effectiveness.

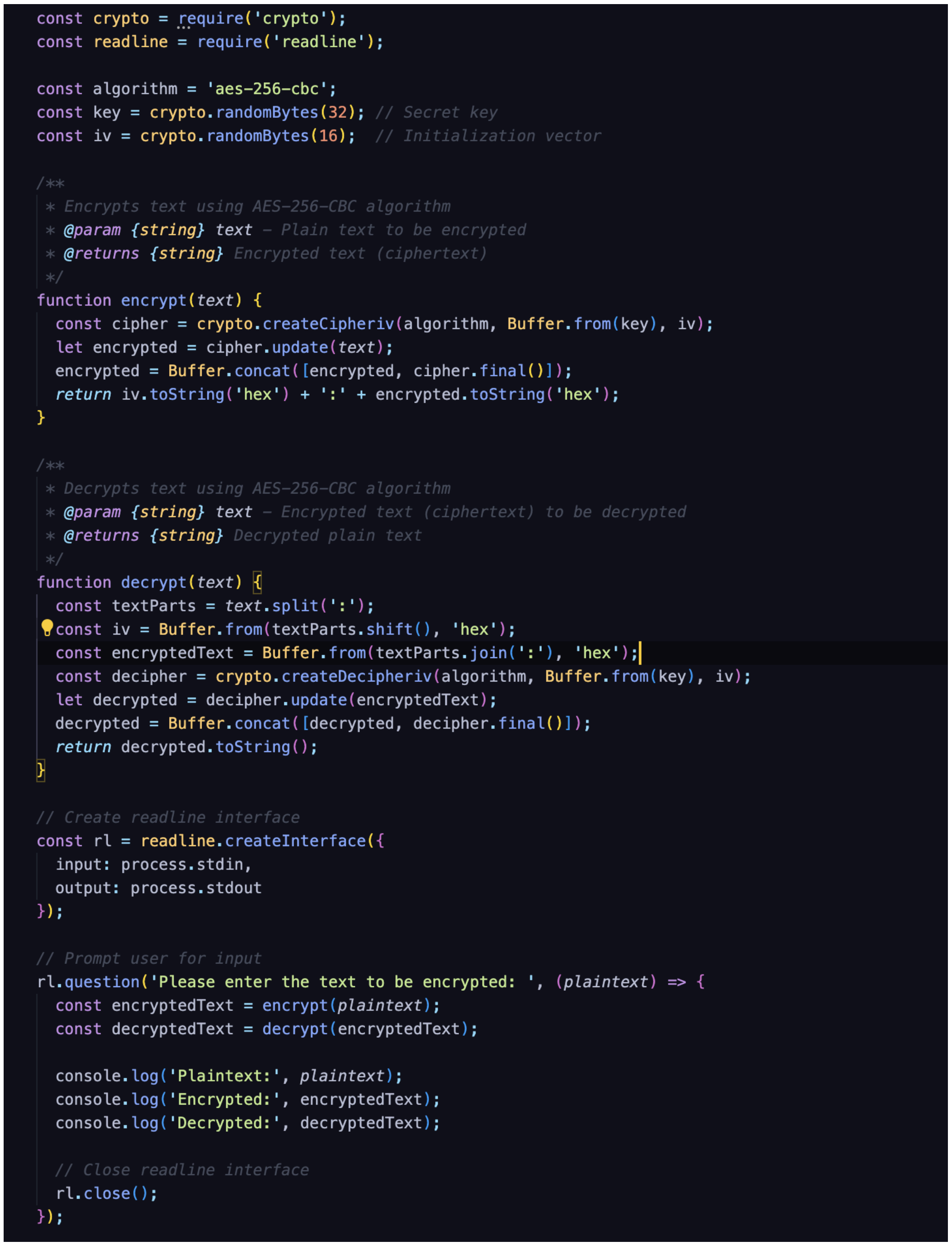
Strengths:- Speed: Blowfish is faster than many other encryption algorithms.- Flexibility: It allows for variable key lengths, providing a balance between security and performance.

Weaknesses:- Key setup: The key setup process is relatively slow compared to other algorithms.- Superseded: Newer algorithms like AES are preferred due to better overall security and standardization.

2. Implement Encryption Mechanisms

Encryption and Decryption Functions

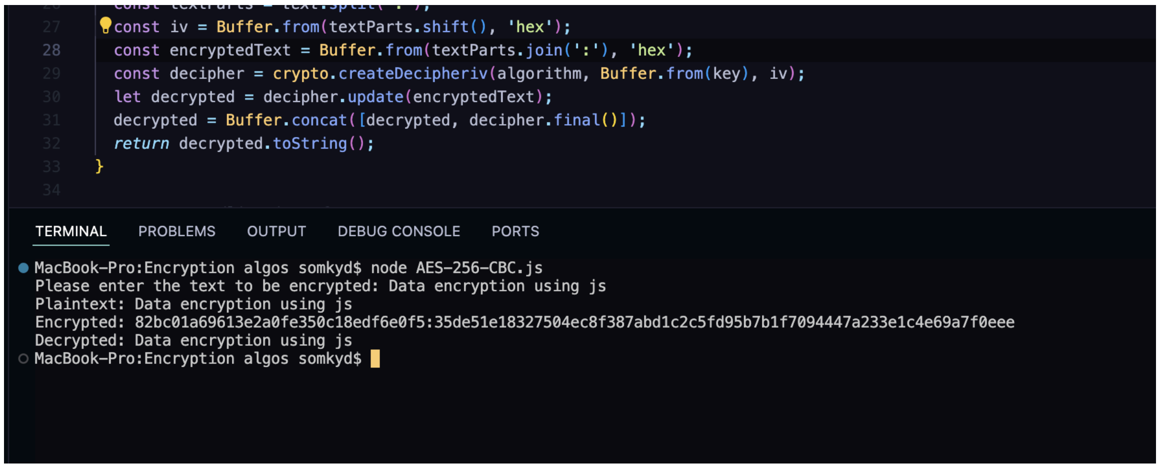
We implemented the encryption and decryption functions using the Node.js `crypto` module. The functions use the AES-256-CBC algorithm for converting plaintext to ciphertext and vice versa.



```javascript
const crypto = require('crypto');
const readline = require('readline');

const algorithm = 'aes-256-cbc';
const key = crypto.randomBytes(32); // Secret key
const iv = crypto.randomBytes(16);  // Initialization vector

/**
 * Encrypts text using AES-256-CBC algorithm
 * @param {string} text - Plain text to be encrypted
 * @returns {string} Encrypted text (ciphertext)
 */
function encrypt(text) {
  const cipher = crypto.createCipheriv(algorithm, Buffer.from(key), iv);
  let encrypted = cipher.update(text);
  encrypted = Buffer.concat([encrypted, cipher.final()]);
  return iv.toString('hex') + ':' + encrypted.toString('hex');
}

/**
 * Decrypts text using AES-256-CBC algorithm
 * @param {string} text - Encrypted text (ciphertext) to be decrypted
 * @returns {string} Decrypted plain text
 */
function decrypt(text) {
  const textParts = text.split(':');
  const iv = Buffer.from(textParts.shift(), 'hex');
  const encryptedText = Buffer.from(textParts.join(':'), 'hex');
  const decipher = crypto.createDecipheriv(algorithm, Buffer.from(key), iv);
  let decrypted = decipher.update(encryptedText);
  decrypted = Buffer.concat([decrypted, decipher.final()]);
  return decrypted.toString();
}

// Create readline interface
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// Prompt user for input
rl.question('Please enter the text to be encrypted: ', (plaintext) => {
  const encryptedText = encrypt(plaintext);
  const decryptedText = decrypt(encryptedText);

  console.log('Plaintext:', plaintext);
  console.log('Encrypted:', encryptedText);
  console.log('Decrypted:', decryptedText);

  // Close readline interface
  rl.close();
});
```

3. Testing the Effectiveness of the Encryption Implementation

We tested the implementation using sample data. The encryption function converts plaintext to ciphertext, and the decryption function converts it back to plaintext, ensuring that the decrypted text matches the original plaintext.



```
    const iv = Buffer.from(textParts.shift(), 'hex');
    const encryptedText = Buffer.from(textParts.join(':'), 'hex');
    const decipher = crypto.createDecipheriv(algorithm, Buffer.from(key), iv);
    let decrypted = decipher.update(encryptedText);
    decrypted = Buffer.concat([decrypted, decipher.final()]);
    return decrypted.toString();
}
```

```
MacBook-Pro:Encryption algos somkyd$ node AES-256-CBC.js
Please enter the text to be encrypted: Data encryption using js
Plaintext: Data encryption using js
Encrypted: 82bc01a69613e2a0fe350c18edf6e0f5:35de51e18327504ec8f387abd1c2c5fd95b7b1f7094447a233e1c4e69a7f0eee
Decrypted: Data encryption using js
MacBook-Pro:Encryption algos somkyd$
```