Step 1: Start

Step 2: Input Key and Message:

2.1:Prompt "Enter key" and store it as key.

2.2:Prompt "Enter a message" and store it as message.

2.3:Calculate len as the length of message.

```
print("Enter key: ")
key = user input
print("Enter a message: ")
message = user input
len = length of message
```

Step 3: Initialize the State Array s:

3.1:Set up the s array of size 256 with values from 0 to 255.

```
for i = 0 to 255:
   s[i] = i
```

Step 4: Key Scheduling Algorithm (KSA):

4.1:Initialize j to 0.

4.2:For each position i in s, calculate j using key, then swap s[i] with s[j].

4.3:Use modulo operations to ensure the values remain within bounds.

```
j = 0
keylen = length of key
for i = 0 to 255:
   j = (j + s[i] + key[i % keylen]) % 256
   swap s[i] and s[j]
```

Step 5: Pseudo-Random Generation Algorithm (PRGA) for Encryption:

5.1:Set i and j to 0.

5.2:For each character n in message:

5.2.1:Update i and j, then swap s[i] with s[j].

5.2.2:XOR the generated keystream value with message[n] to get ciphertext[n].

```
i = 0
```

j = 0

for n = 0 to len-1:

i = (i + 1) % 256

j = (j + s[i]) % 256

swap s[i] and s[j]

ciphertext[n] = s[(s[i] + s[j]) % 256] XOR message[n]

Step 6: Display Ciphertext in Hexadecimal:

6.1:Print each byte in ciphertext as a two-digit hexadecimal value.

print("Ciphertext (hex): ")

for i = 0 to len-1:

print ciphertext[i] as two-digit hex

Step 7: Reinitialize State Array s for Decryption:

7.1:Reset s and perform the Key Scheduling Algorithm (KSA) again.

for i = 0 to 255:

s[i] = i

repeat KSA steps

Step 8: PRGA for Decryption:

8.1:Using the same procedure as encryption, XOR each keystream value with ciphertext[n] to retrieve plaintext[n].

i = (i + 1) % 256

j = (j + s[i]) % 256

swap s[i] and s[j]

plaintext[n] = s[(s[i] + s[j]) % 256] XOR ciphertext[n]

Step 9: Display Plaintext:

9.1:Print the resulting plaintext.

print("Plaintext is: ", plaintext)

Step 10: Stop