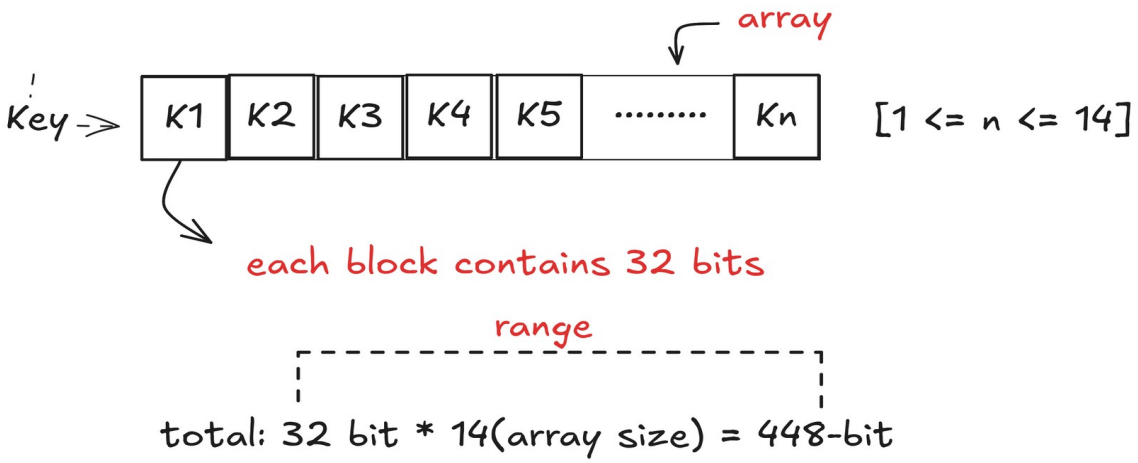
Introduction to Blowfish Encryption

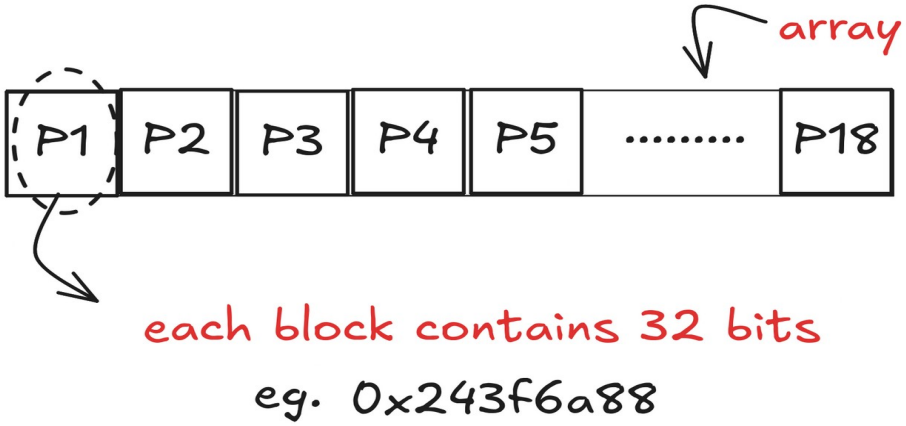
Blowfish is a symmetric key block cipher that encrypts data in 64-bit blocks using a variable key size from 32 to 448 bits. It employs a Feistel network structure with multiple rounds of substitution and permutation operations.

How the Blowfish Works:

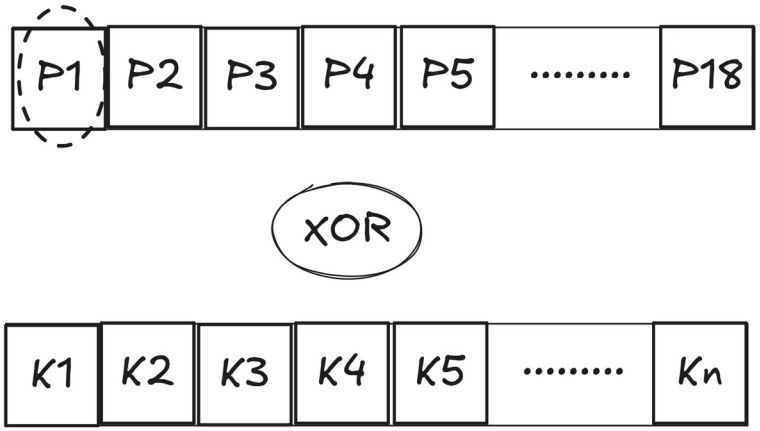
Input for each block: 64 bits.
Key size: Variable, from 32 to 448 bits.



Initialize an array P:



Generation of sub-keys



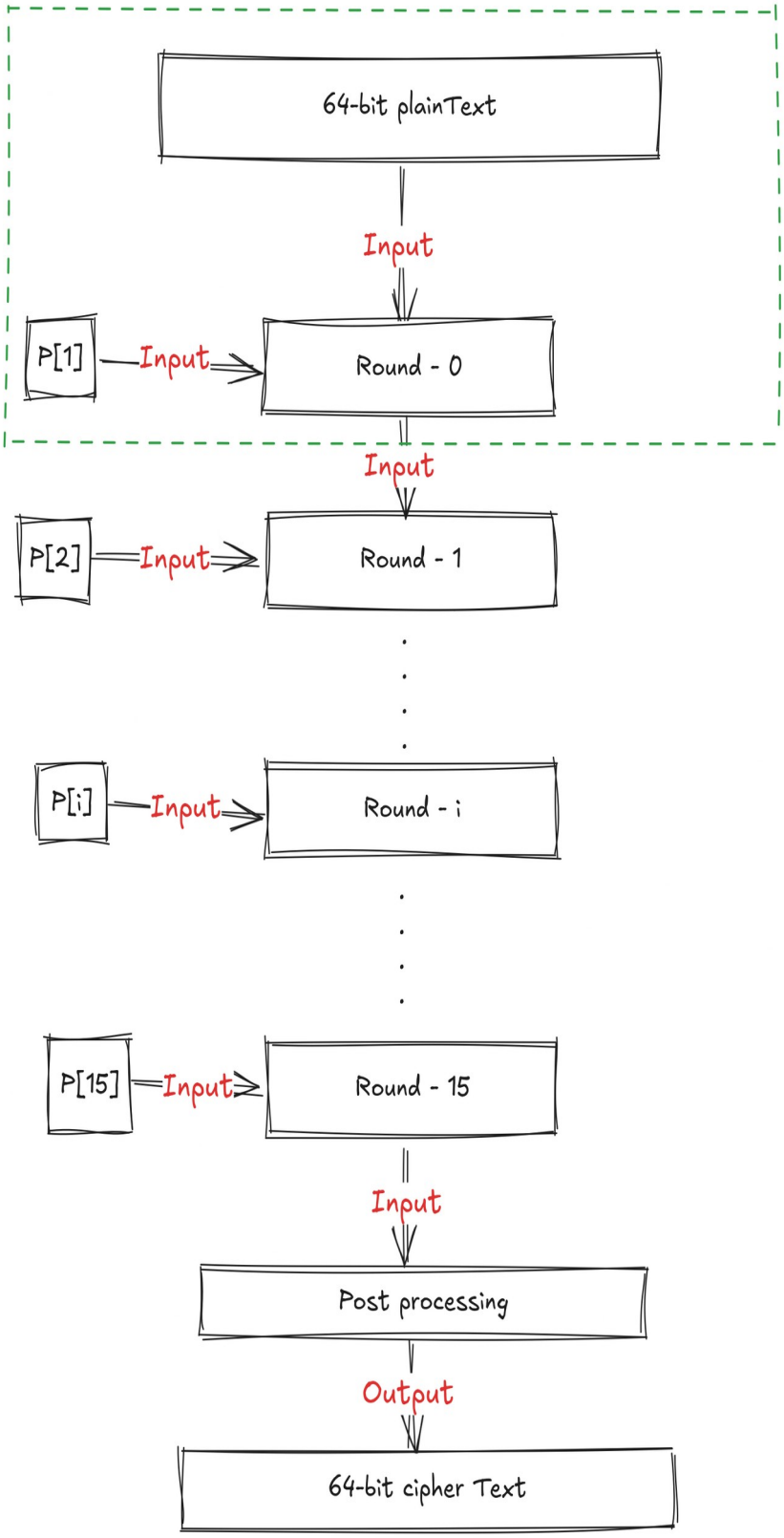

```
P1= P1 xor K1
P2= P2 xor K2
.
.
.
P14= P14 xor K14
P15= P15 xor K1
P16= P16 xor K2
P17= P17 xor K3
P18= P18 xor K4
```

```
p[0] = 0x243f6a88; p[9] = 0xb7d4a57d;
p[1] = 0x85a308d3; p[10] = 0x3708a35d;
p[2] = 0x1319f8a2; p[11] = 0x7f4f5b43;
p[3] = 0x6d9d4a3b; p[12] = 0x0a831a55;
p[4] = 0x6d4d34c0; p[13] = 0xb68e4c10;
p[5] = 0x00e6f4f0; p[14] = 0x4e4d1c7d;
p[6] = 0x701d7cb6; p[15] = 0x66d3ab76;
p[7] = 0xc1a4b9c3; p[16] = 0xf8566b42;
p[8] = 0x4632e00d; p[17] = 0x2c4b55b3;
```

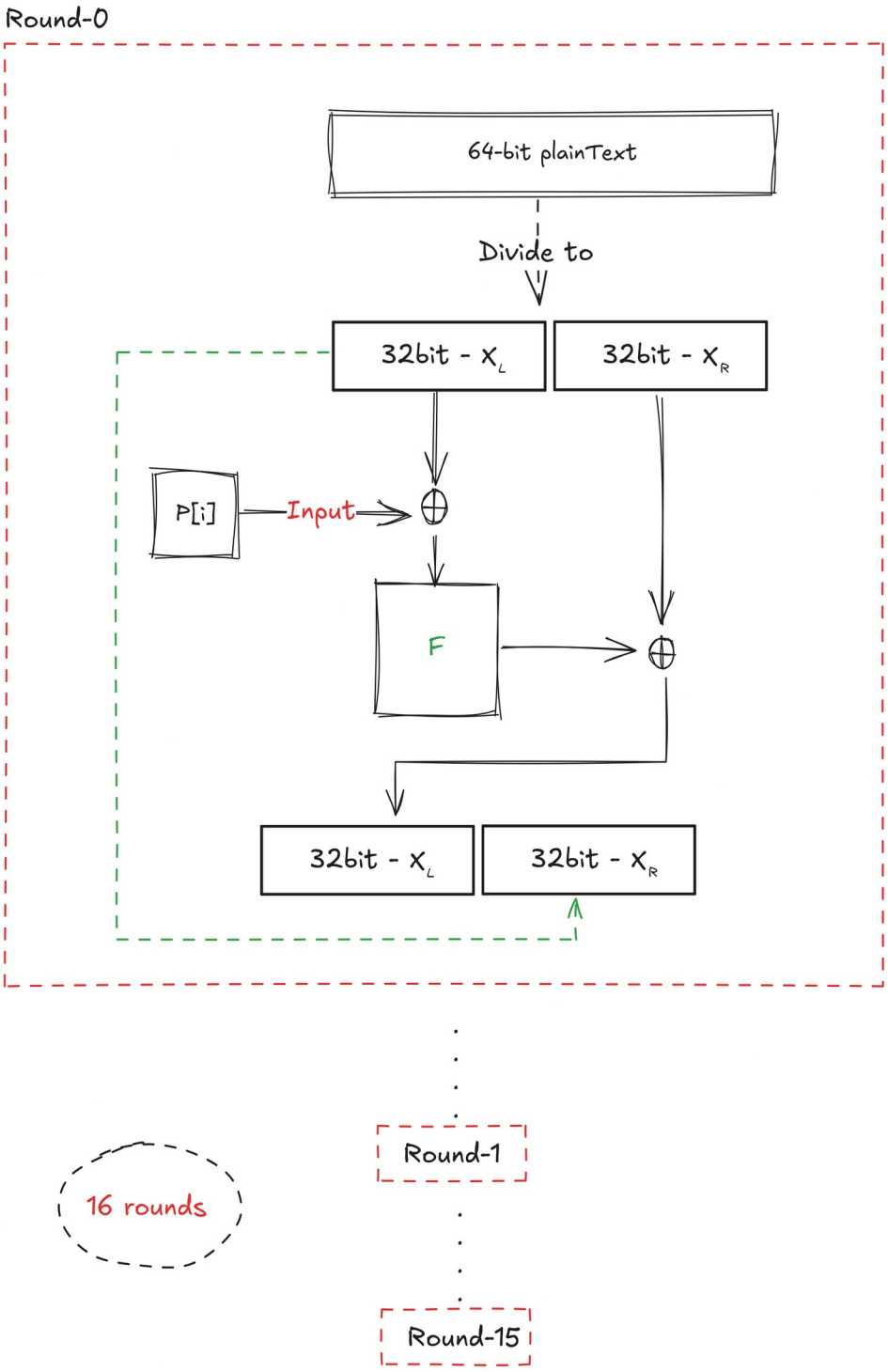
initialize S-Boxes

```
const uint32_t S[4][256] = {
    {
        0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96, 0xba7c9045, 0x2de92c6f,
        0x03c5d95d, 0x6e2b9b0f, 0x4a7484aa, 0x10e3f0c8, 0x63e6d6b4, 0x91c8b7f4, 0x31a8a1b5, 0x1b6d6c72,
        0x96a1d2b8, 0x00d0a8d5, 0x35d86a7f, 0x9d4f1a07, 0x5b9c3f43, 0x2a8a2a2b, 0x5d03771e, 0x1d968009,
        0x6b4894b8, 0x0d6b2800, 0x3728e0d4, 0xa7d8c536, 0x2a1d8f6e, 0xb4e91f5f, 0x0f2e3b6f, 0x14a8d377,
        0x94f94530, 0x0a578048, 0x1c76b9b0, 0x18a867ad, 0x91d989e8, 0x1a0f7e93, 0x9a5b59c0, 0x37f4f70b,
        0xd6d3c38c, 0x4694e60f, 0x292b2d3b, 0x64f71bb7, 0xa44fcd4e, 0x9c660b54, 0x6712aef4, 0x07c839bb,
        0x61e2f7e6, 0x8d56b8c4, 0x715d43a1, 0x2b1f45f4, 0x2e4efb3f, 0x87b3d845, 0xb6702d88, 0x3b6d9c77,
        0x22e65951, 0x90a8a541, 0xaee519d6, 0x371cb1e4, 0x5628e4fc, 0xe24c45da, 0xe1e9bdc6, 0xf14d8b56,
        0x9c6b48f5, 0x28f3f4b5, 0x15671d83, 0x038e1d63, 0x4e9c2b1e, 0x6936f92b, 0x48a74513, 0x80e0db29,
        0x6a393a37, 0x50f75542, 0x62994b4b, 0xbfd04463, 0x8ac0a715, 0x37148942, 0x17b19c4a, 0x06c83258,
        0x146f5bd1, 0xa407a5d7, 0x0832826f, 0xf5b1c1c7, 0x3e2878ff, 0x81e3d4b1, 0x8b76a445, 0x26a60551,
        0x5c45b0e6, 0x4697b9f8, 0x7c4b68c3, 0x0d7b5b77, 0x6d70e98a, 0x28b418fb, 0x60bfc8d6, 0x350388b4,
        0x56c0447b, 0xa54908de, 0x86baf973, 0x5c9f25d8, 0x9eec737b, 0x3e0dc787, 0x7e957c58, 0x6c9d7dcf,
        0x54a44572, 0x5ab7e932, 0x305a2f62, 0x84a1d36e, 0x1e3dc40a, 0x7e6f0cb0, 0xa0d28f67, 0x13d31cfc,
        0x5a407e93, 0x302c5076, 0x7df07b54, 0x9bc4d70d, 0x16d0e5b8, 0x9e5b0574, 0x2869c2d3, 0x72bda5cb,
        0xb21e1043, 0x96c6c0f2, 0x04629038, 0x4d85a07f, 0xb04411b8, 0x76e2a468, 0x1b837cb0, 0x5b243b3e
    },
    {
        0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96, 0xba7c9045, 0x2de92c6f,
        0x03c5d95d, 0x6e2b9b0f, 0x4a7484aa, 0x10e3f0c8, 0x63e6d6b4, 0x91c8b7f4, 0x31a8a1b5, 0x1b6d6c72,
        0x96a1d2b8, 0x00d0a8d5, 0x35d86a7f, 0x9d4f1a07, 0x5b9c3f43, 0x2a8a2a2b, 0x5d03771e, 0x1d968009,
        0x6b4894b8, 0x0d6b2800, 0x3728e0d4, 0xa7d8c536, 0x2a1d8f6e, 0xb4e91f5f, 0x0f2e3b6f, 0x14a8d377,
        0x94f94530, 0x0a578048, 0x1c76b9b0, 0x18a867ad, 0x91d989e8, 0x1a0f7e93, 0x9a5b59c0, 0x37f4f70b,
        0xd6d3c38c, 0x4694e60f, 0x292b2d3b, 0x64f71bb7, 0xa44fcd4e, 0x9c660b54, 0x6712aef4, 0x07c839bb,
        0x61e2f7e6, 0x8d56b8c4, 0x715d43a1, 0x2b1f45f4, 0x2e4efb3f, 0x87b3d845, 0xb6702d88, 0x3b6d9c77,
        0x22e65951, 0x90a8a541, 0xaee519d6, 0x371cb1e4, 0x5628e4fc, 0xe24c45da, 0xe1e9bdc6, 0xf14d8b56,
        0x9c6b48f5, 0x28f3f4b5, 0x15671d83, 0x038e1d63, 0x4e9c2b1e, 0x6936f92b, 0x48a74513, 0x80e0db29,
        0x6a393a37, 0x50f75542, 0x62994b4b, 0xbfd04463, 0x8ac0a715, 0x37148942, 0x17b19c4a, 0x06c83258,
        0x146f5bd1, 0xa407a5d7, 0x0832826f, 0xf5b1c1c7, 0x3e2878ff, 0x81e3d4b1, 0x8b76a445, 0x26a60551,
        0x5c45b0e6, 0x4697b9f8, 0x7c4b68c3, 0x0d7b5b77, 0x6d70e98a, 0x28b418fb, 0x60bfc8d6, 0x350388b4,
        0x56c0447b, 0xa54908de, 0x86baf973, 0x5c9f25d8, 0x9eec737b, 0x3e0dc787, 0x7e957c58, 0x6c9d7dcf,
        0x54a44572, 0x5ab7e932, 0x305a2f62, 0x84a1d36e, 0x1e3dc40a, 0x7e6f0cb0, 0xa0d28f67, 0x13d31cfc,
        0x5a407e93, 0x302c5076, 0x7df07b54, 0x9bc4d70d, 0x16d0e5b8, 0x9e5b0574, 0x2869c2d3, 0x72bda5cb,
        0xb21e1043, 0x96c6c0f2, 0x04629038, 0x4d85a07f, 0xb04411b8, 0x76e2a468, 0x1b837cb0, 0x5b243b3e
    },
    {
        0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96, 0xba7c9045, 0x2de92c6f,
        0x03c5d95d, 0x6e2b9b0f, 0x4a7484aa, 0x10e3f0c8, 0x63e6d6b4, 0x91c8b7f4, 0x31a8a1b5, 0x1b6d6c72,
        0x96a1d2b8, 0x00d0a8d5, 0x35d86a7f, 0x9d4f1a07, 0x5b9c3f43, 0x2a8a2a2b, 0x5d03771e, 0x1d968009,
        0x6b4894b8, 0x0d6b2800, 0x3728e0d4, 0xa7d8c536, 0x2a1d8f6e, 0xb4e91f5f, 0x0f2e3b6f, 0x14a8d377,
        0x94f94530, 0x0a578048, 0x1c76b9b0, 0x18a867ad, 0x91d989e8, 0x1a0f7e93, 0x9a5b59c0, 0x37f4f70b,
        0xd6d3c38c, 0x4694e60f, 0x292b2d3b, 0x64f71bb7, 0xa44fcd4e, 0x9c660b54, 0x6712aef4, 0x07c839bb,
        0x61e2f7e6, 0x8d56b8c4, 0x715d43a1, 0x2b1f45f4, 0x2e4efb3f, 0x87b3d845, 0xb6702d88, 0x3b6d9c77,
        0x22e65951, 0x90a8a541, 0xaee519d6, 0x371cb1e4, 0x5628e4fc, 0xe24c45da, 0xe1e9bdc6, 0xf14d8b56,
        0x9c6b48f5, 0x28f3f4b5, 0x15671d83, 0x038e1d63, 0x4e9c2b1e, 0x6936f92b, 0x48a74513, 0x80e0db29,
        0x6a393a37, 0x50f75542, 0x62994b4b, 0xbfd04463, 0x8ac0a715, 0x37148942, 0x17b19c4a, 0x06c83258,
        0x146f5bd1, 0xa407a5d7, 0x0832826f, 0xf5b1c1c7, 0x3e2878ff, 0x81e3d4b1, 0x8b76a445, 0x26a60551,
        0x5c45b0e6, 0x4697b9f8, 0x7c4b68c3, 0x0d7b5b77, 0x6d70e98a, 0x28b418fb, 0x60bfc8d6, 0x350388b4,
        0x56c0447b, 0xa54908de, 0x86baf973, 0x5c9f25d8, 0x9eec737b, 0x3e0dc787, 0x7e957c58, 0x6c9d7dcf,
        0x54a44572, 0x5ab7e932, 0x305a2f62, 0x84a1d36e, 0x1e3dc40a, 0x7e6f0cb0, 0xa0d28f67, 0x13d31cfc,
        0x5a407e93, 0x302c5076, 0x7df07b54, 0x9bc4d70d, 0x16d0e5b8, 0x9e5b0574, 0x2869c2d3, 0x72bda5cb,
        0xb21e1043, 0x96c6c0f2, 0x04629038, 0x4d85a07f, 0xb04411b8, 0x76e2a468, 0x1b837cb0, 0x5b243b3e
    },
    {
        0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96, 0xba7c9045, 0x2de92c6f,
        0x03c5d95d, 0x6e2b9b0f, 0x4a7484aa, 0x10e3f0c8, 0x63e6d6b4, 0x91c8b7f4, 0x31a8a1b5, 0x1b6d6c72,
        0x96a1d2b8, 0x00d0a8d5, 0x35d86a7f, 0x9d4f1a07, 0x5b9c3f43, 0x2a8a2a2b, 0x5d03771e, 0x1d968009,
        0x6b4894b8, 0x0d6b2800, 0x3728e0d4, 0xa7d8c536, 0x2a1d8f6e, 0xb4e91f5f, 0x0f2e3b6f, 0x14a8d377,
        0x94f94530, 0x0a578048, 0x1c76b9b0, 0x18a867ad, 0x91d989e8, 0x1a0f7e93, 0x9a5b59c0, 0x37f4f70b,
        0xd6d3c38c, 0x4694e60f, 0x292b2d3b, 0x64f71bb7, 0xa44fcd4e, 0x9c660b54, 0x6712aef4, 0x07c839bb,
        0x61e2f7e6, 0x8d56b8c4, 0x715d43a1, 0x2b1f45f4, 0x2e4efb3f, 0x87b3d845, 0xb6702d88, 0x3b6d9c77,
        0x22e65951, 0x90a8a541, 0xaee519d6, 0x371cb1e4, 0x5628e4fc, 0xe24c45da, 0xe1e9bdc6, 0xf14d8b56,
        0x9c6b48f5, 0x28f3f4b5, 0x15671d83, 0x038e1d63, 0x4e9c2b1e, 0x6936f92b, 0x48a74513, 0x80e0db29,
        0x6a393a37, 0x50f75542, 0x62994b4b, 0xbfd04463, 0x8ac0a715, 0x37148942, 0x17b19c4a, 0x06c83258,
        0x146f5bd1, 0xa407a5d7, 0x0832826f, 0xf5b1c1c7, 0x3e2878ff, 0x81e3d4b1, 0x8b76a445, 0x26a60551,
        0x5c45b0e6, 0x4697b9f8, 0x7c4b68c3, 0x0d7b5b77, 0x6d70e98a, 0x28b418fb, 0x60bfc8d6, 0x350388b4,
        0x56c0447b, 0xa54908de, 0x86baf973, 0x5c9f25d8, 0x9eec737b, 0x3e0dc787, 0x7e957c58, 0x6c9d7dcf,
        0x54a44572, 0x5ab7e932, 0x305a2f62, 0x84a1d36e, 0x1e3dc40a, 0x7e6f0cb0, 0xa0d28f67, 0x13d31cfc,
        0x5a407e93, 0x302c5076, 0x7df07b54, 0x9bc4d70d, 0x16d0e5b8, 0x9e5b0574, 0x2869c2d3, 0x72bda5cb,
        0xb21e1043, 0x96c6c0f2, 0x04629038, 0x4d85a07f, 0xb04411b8, 0x76e2a468, 0x1b837cb0, 0x5b243b3e
    }
};
```

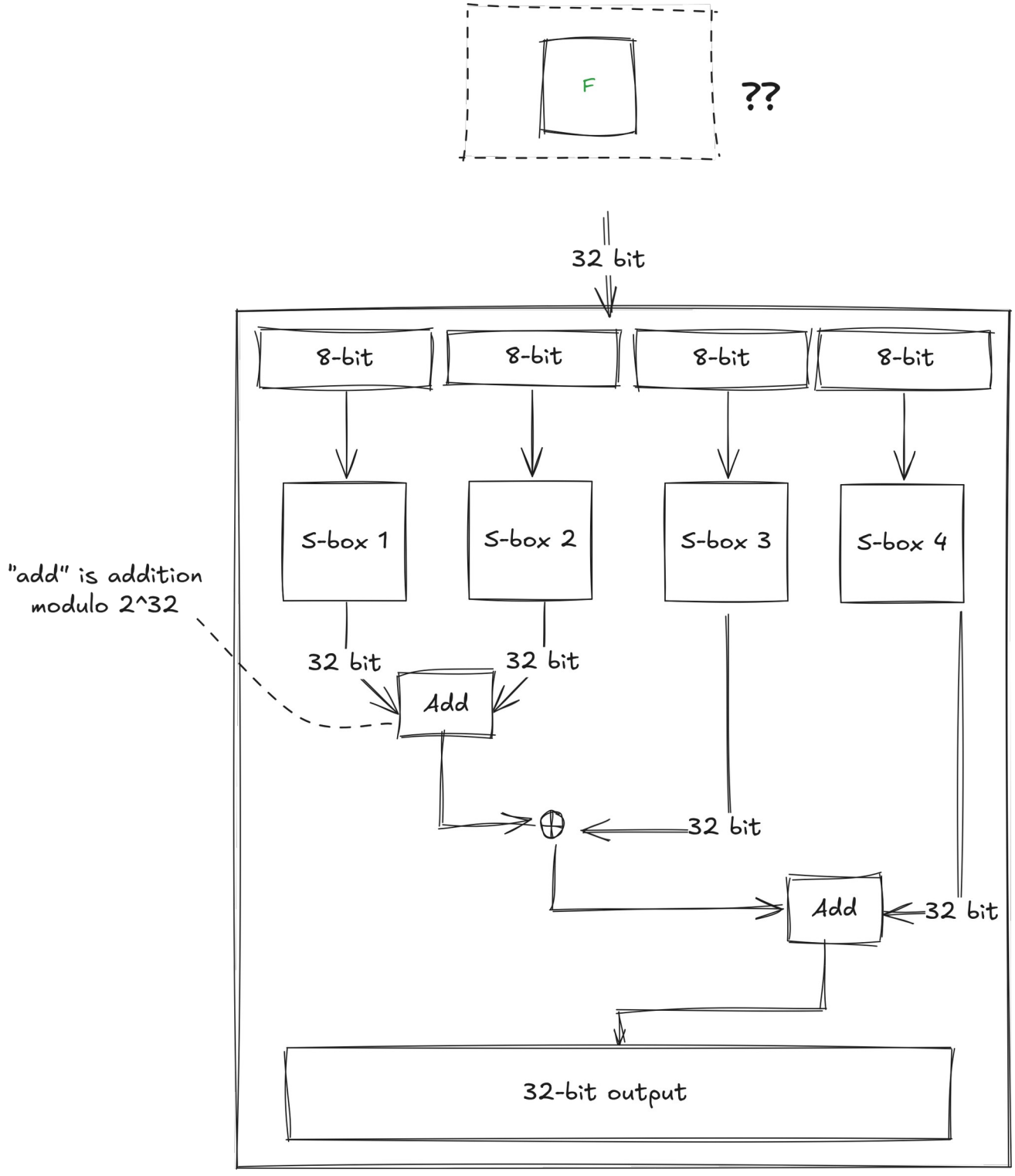
Encryption


64-bit plainText
Input
P[1]
Input
Round - 0
Input
P[2]
Input
Round - 1
P[i]
Input
Round - i
P[15]
Input
Round - 15
Input
Post processing
Output
64-bit cipher Text

Delve deeper inside the round

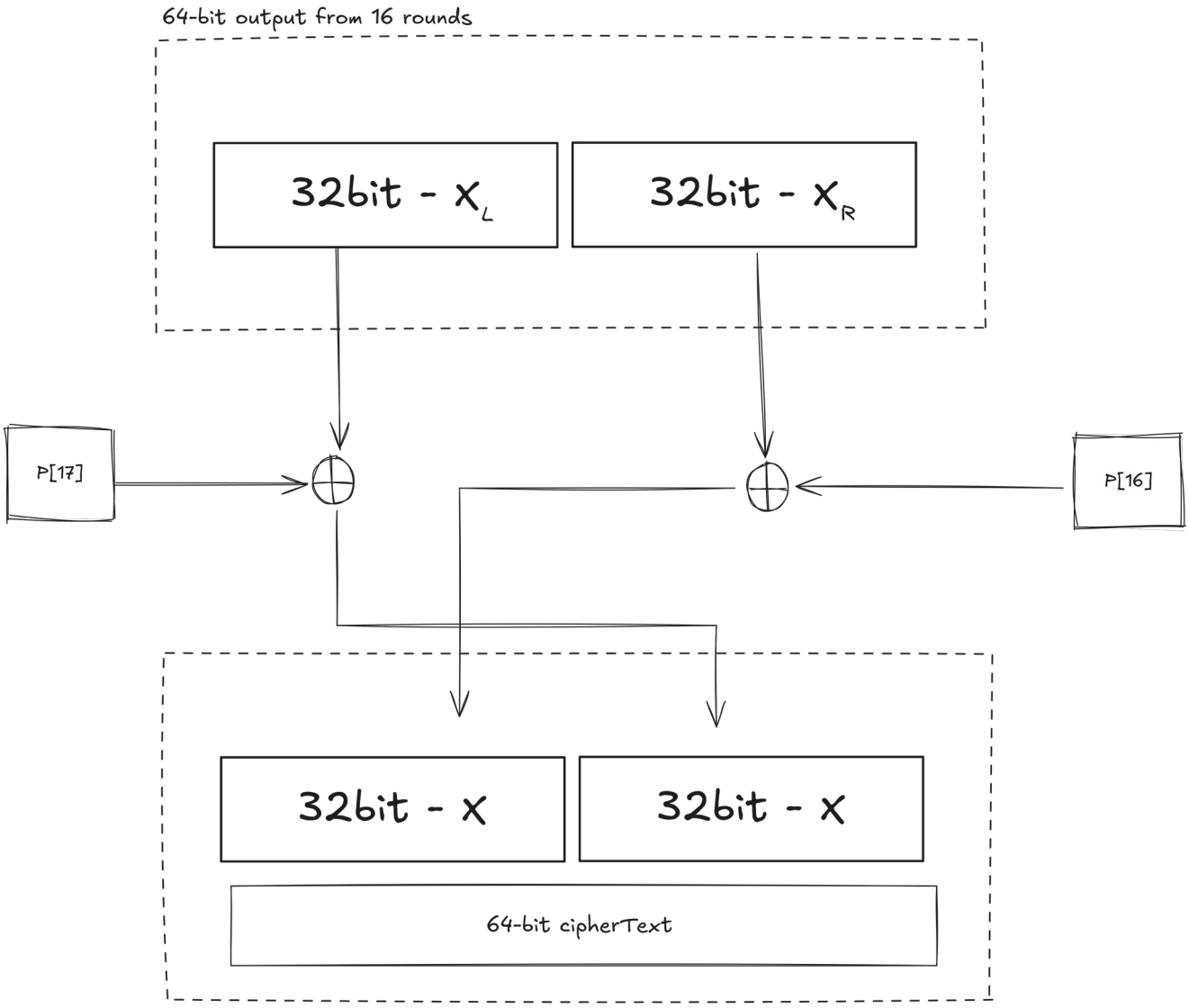


what is Function F?



post-processing

After the 16 rounds are completed, we have a left 32-bit and a right 32-bit.



The Triple-key Blowfish

How Triple-key Blowfish Works

Key Setup:

- **Three Keys:** `key1`, `key2`, and `key3`. Each key is used for a different stage of the encryption and decryption process.
- **Key Length**: Each key can be up to 448 bits long;

Encryption Process:

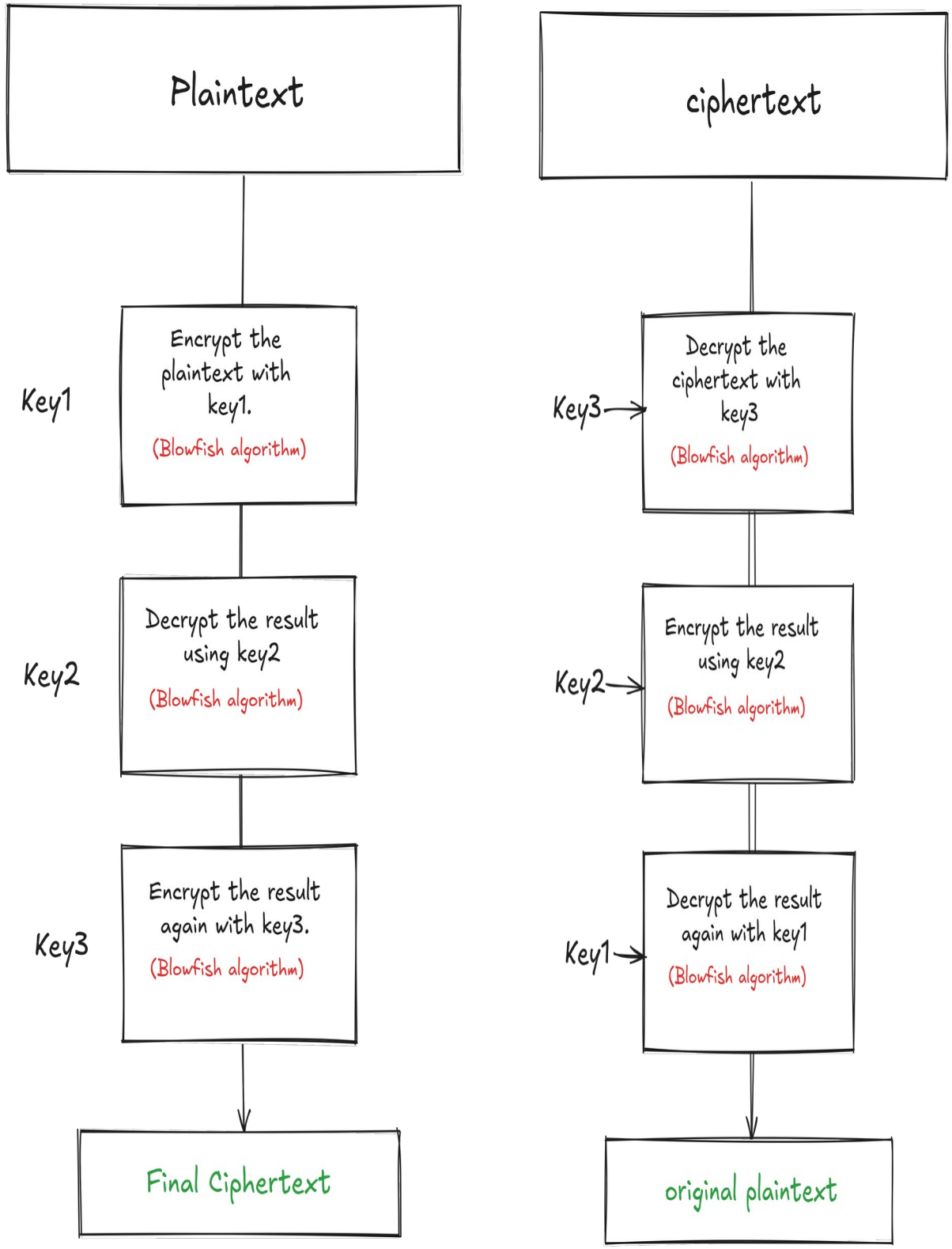
- **Initial Encryption:** Encrypt the plaintext with key1.
- **Decryption:** Decrypt the result using `key2`.
- **Final Encryption:** Encrypt the result again with `key3`.

Decryption Process:

- **Initial Decryption**: Decrypt the ciphertext with `key3`.
- **Encryption**: Encrypt the result using `key2`.
- **Final Decryption**: Decrypt the result again with `key1`.

Wrap up:

Encryption Process:	**Decryption Process:**
Input: Plaintext data. **Step 1**: Encrypt with `key1`. **Step 2**: Decrypt the intermediate result with `key2`. **Step 3**: Encrypt the final result with `key3`. **Output**: Encrypted ciphertext.	**Input: Ciphertext data.** • **Step 1**: Decrypt with `key3`. • **Step 2**: Encrypt the intermediate result with `key2`. • **Step 3**: Decrypt the final result with `key1`. • **Output**: Decrypted plaintext.


Plaintext
Encrypt the plaintext with key1.
(Blowfish algorithm)
Key1
Decrypt the result using key2
(Blowfish algorithm)
Key2
Encrypt the result again with key3.
(Blowfish algorithm)
Key3
Final Ciphertext
ciphertext
Decrypt the ciphertext with key3
(Blowfish algorithm)
Key3
Encrypt the result using key2
(Blowfish algorithm)
Key2
Decrypt the result again with key1
(Blowfish algorithm)
Key1
original plaintext

Benchmarking

:~$ python3 benchmark.py

The script will perform benchmarks for the following algorithms and output the results:

CBC Mode Encryption and Decryption

Benchmarking encrypt_cbc

10000 random bytes in 0.01792 sec
10000 random bytes in 0.01897 sec
10000 random bytes in 0.01687 sec
10000 random bytes in 0.01935 sec
10000 random bytes in 0.02697 sec
10000 random bytes in 0.02001 sec (average)

Benchmarking decrypt_cbc:

10000 random bytes in 0.02177 sec
10000 random bytes in 0.02155 sec
10000 random bytes in 0.02133 sec
10000 random bytes in 0.01915 sec
10000 random bytes in 0.01824 sec
10000 random bytes in 0.02041 sec (average)

Triple-Key Blowfish Encryption and Decryption

Benchmarking encrypt_text_triple_blowfish:

10000 random bytes in 0.08953 sec
10000 random bytes in 0.06500 sec
10000 random bytes in 0.06526 sec
10000 random bytes in 0.05713 sec
10000 random bytes in 0.04344 sec
10000 random bytes in 0.06407 sec (average)

Benchmarking decrypt_text_triple_blowfish

10000 random bytes in 0.05065 sec
10000 random bytes in 0.07105 sec
10000 random bytes in 0.05591 sec
10000 random bytes in 0.05827 sec
10000 random bytes in 0.06196 sec
10000 random bytes in 0.05957 sec (average)

Benchmarking encrypt_image_triple_blowfish

Encrypted image saved to encrypted_fish.jpg
Image 1024 * 793 in 0.63568 sec
Encrypted image saved to encrypted_fish.jpg
Image 1024 * 793 in 0.66269 sec
Encrypted image saved to encrypted_fish.jpg
Image 1024 * 793 in 0.64372 sec
Encrypted image saved to encrypted_fish.jpg
Image 1024 * 793 in 0.62553 sec
Encrypted image saved to encrypted_fish.jpg
Image 1024 * 793 in 0.60770 sec
Image 1024 * 793 in 0.63506 sec (average)

Benchmarking decrypt_image_triple_blowfish

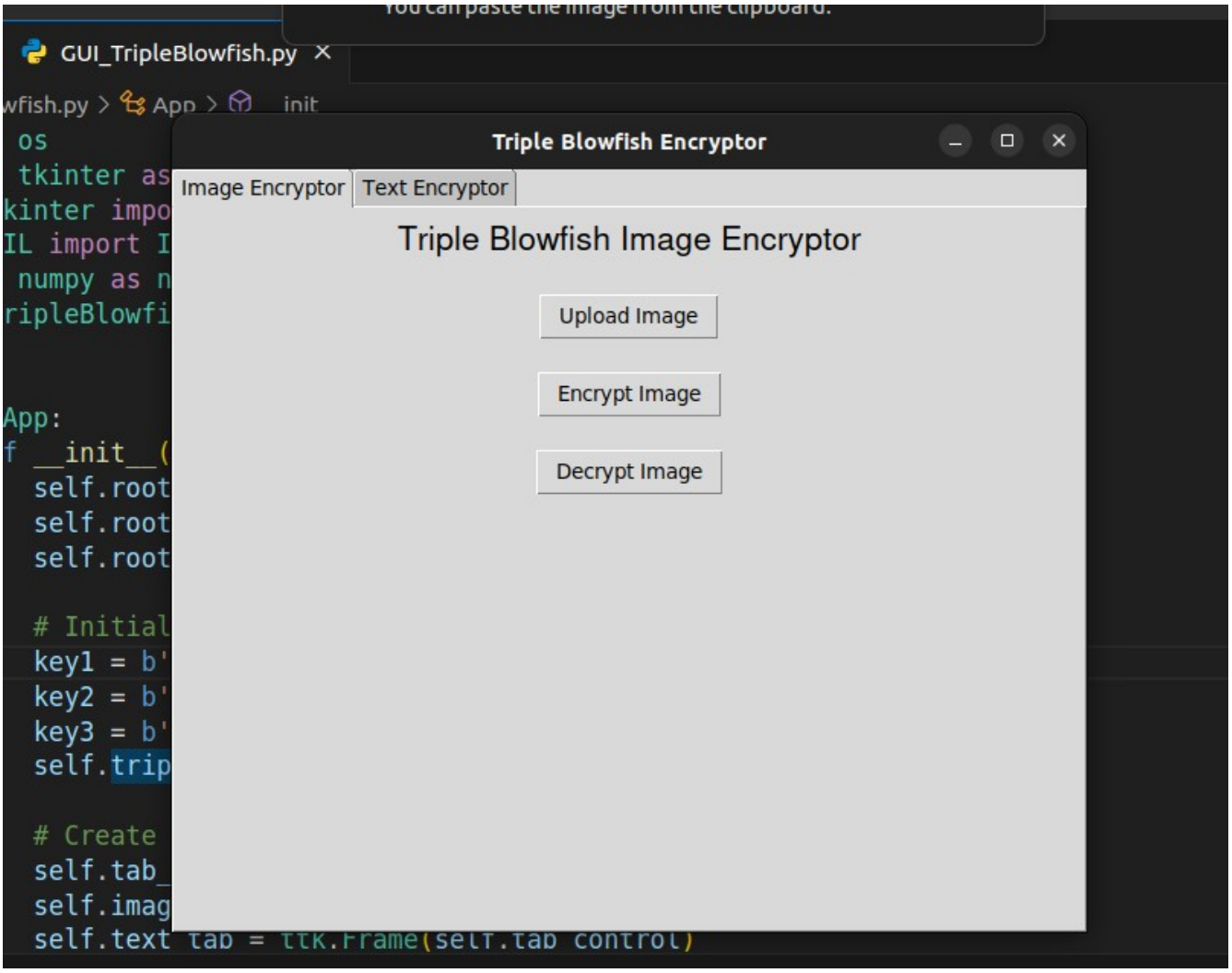
Decrypted image saved to decrypted_fish.jpg
Image 1024 * 793 in 0.60871 sec
Decrypted image saved to decrypted_fish.jpg
Image 1024 * 793 in 0.57858 sec
Decrypted image saved to decrypted_fish.jpg
Image 1024 * 793 in 0.62017 sec
Decrypted image saved to decrypted_fish.jpg
Image 1024 * 793 in 0.64028 sec
Decrypted image saved to decrypted_fish.jpg
Image 1024 * 793 in 0.65398 sec
Image 1024 * 793 in 0.62034 sec (average)

Summary

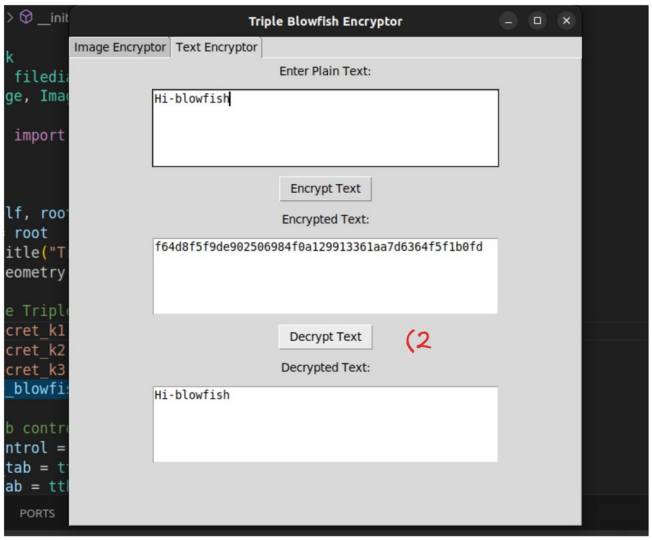
The benchmark.py script tests all the algorithms six times and provides the average time taken for each operation.

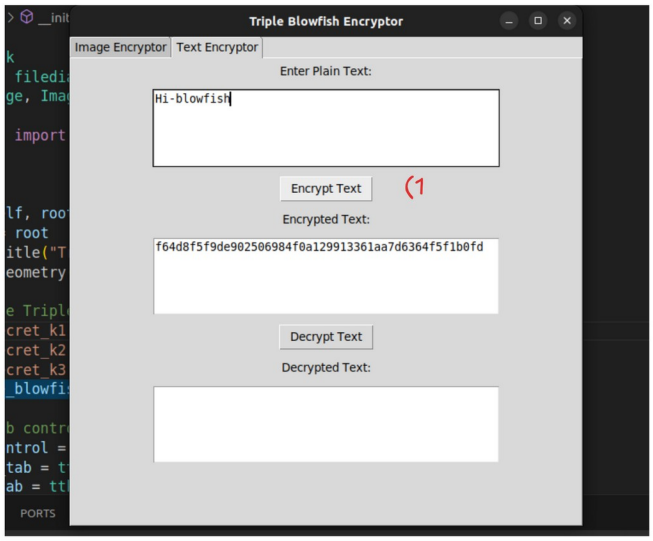
Triple blowfish - GUI (Test)

When you run the application using `python3 GUI_TripleBlowfish.py`, you will see options in the navigation bar allowing you to choose between text or image encryption."

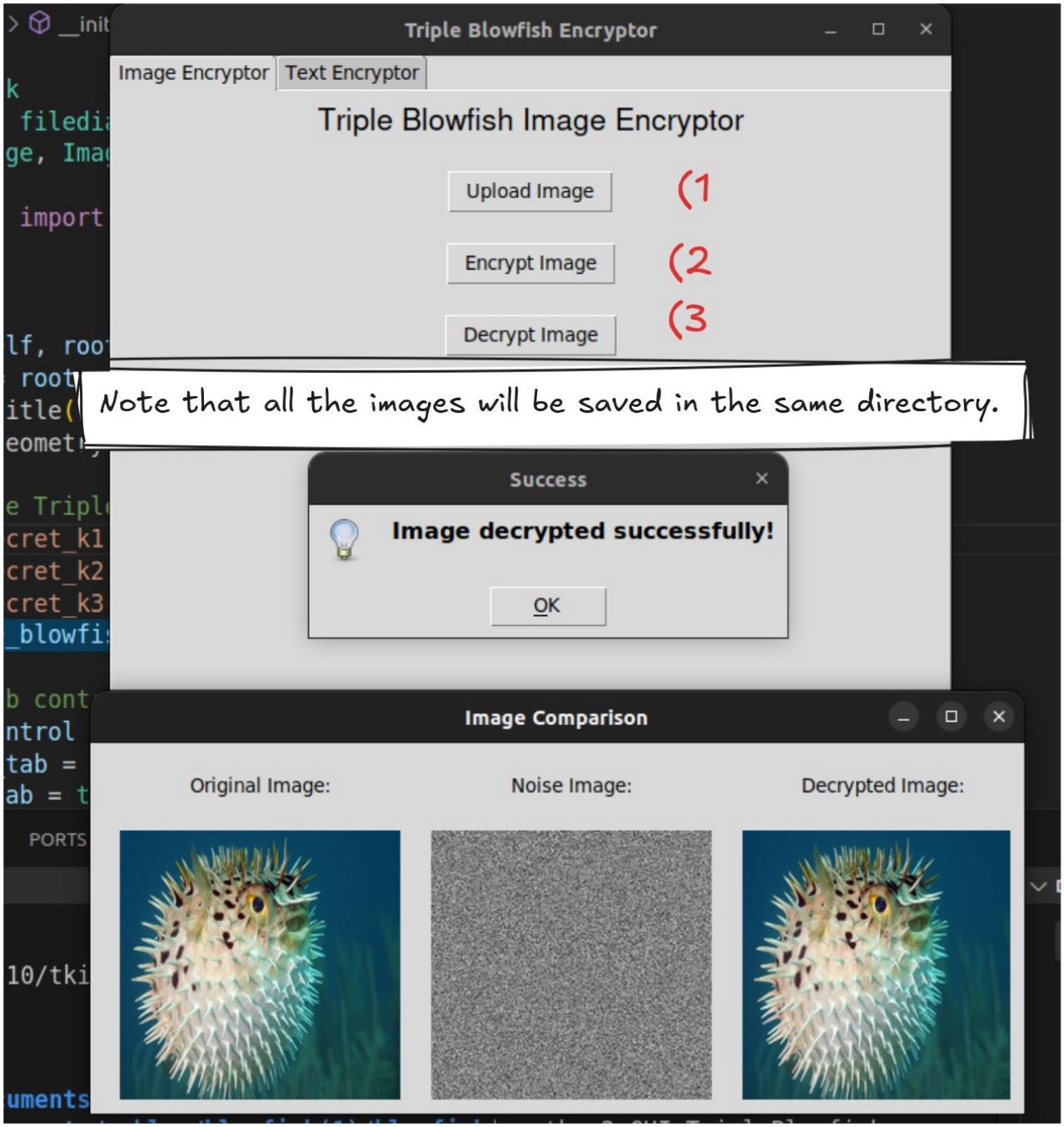


1 - Text


Triple Blowfish Encryptor
Image Encryptor
Text Encryptor
Enter Plain Text:
Hi-blowfish
Encrypt Text
Encrypted Text:
f64d8f5f9de902506984f0a129913361aa7d6364f5f1b0fd
Decrypt Text
(2
Decrypted Text:
Hi-blowfish


Triple Blowfish Encryptor
Image Encryptor
Text Encryptor
Enter Plain Text:
Hi-blowfish
Encrypt Text
(1
Encrypted Text:
f64d8f5f9de902506984f0a129913361aa7d6364f5f1b0fd
Decrypt Text
Decrypted Text:

2 – Image


Triple Blowfish Encryptor
Image Encryptor
Text Encryptor
Triple Blowfish Image Encryptor
Upload Image
(1
Encrypt Image
(2
Decrypt Image
(3
Note that all the images will be saved in the same directory.
Success
Image decrypted successfully!
OK
Image Comparison
Original Image:
Noise Image:
Decrypted Image: