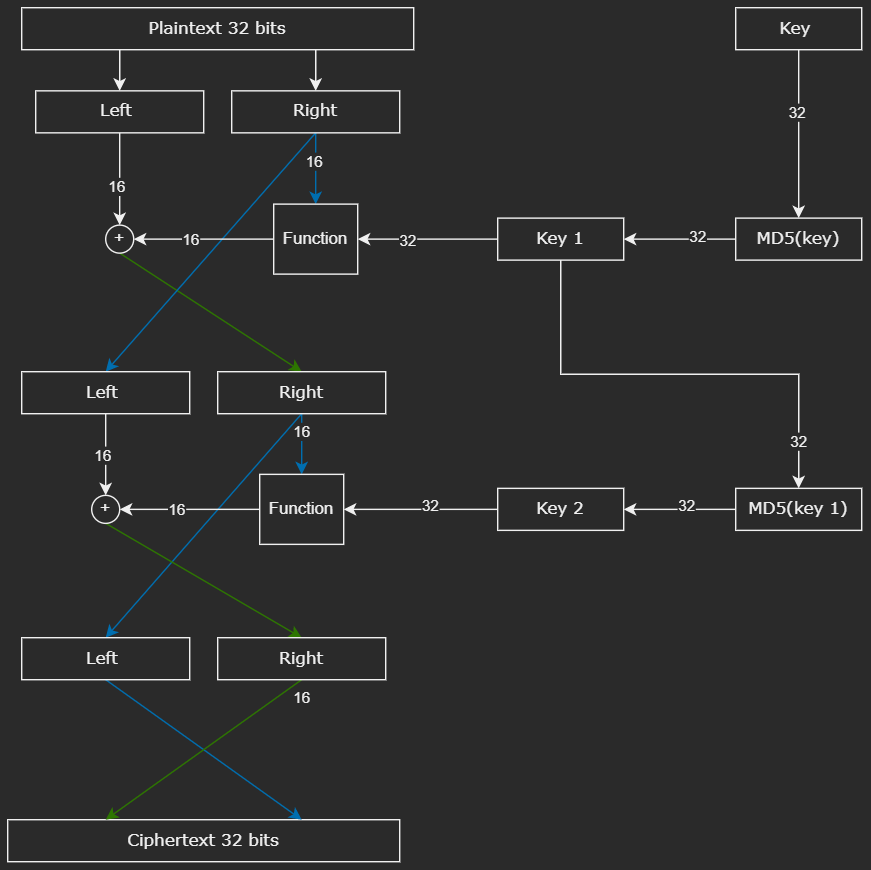
**Encrypt.java structure:**



The plain text is divided into two 16-bits blocks called Left and Right.

The key is hashed using MD5 every round, to increase key-plaintext relationship ambiguity. In addition, the FUNCTION uses a 16 bits permutation table (p-box) to further increase complexity.

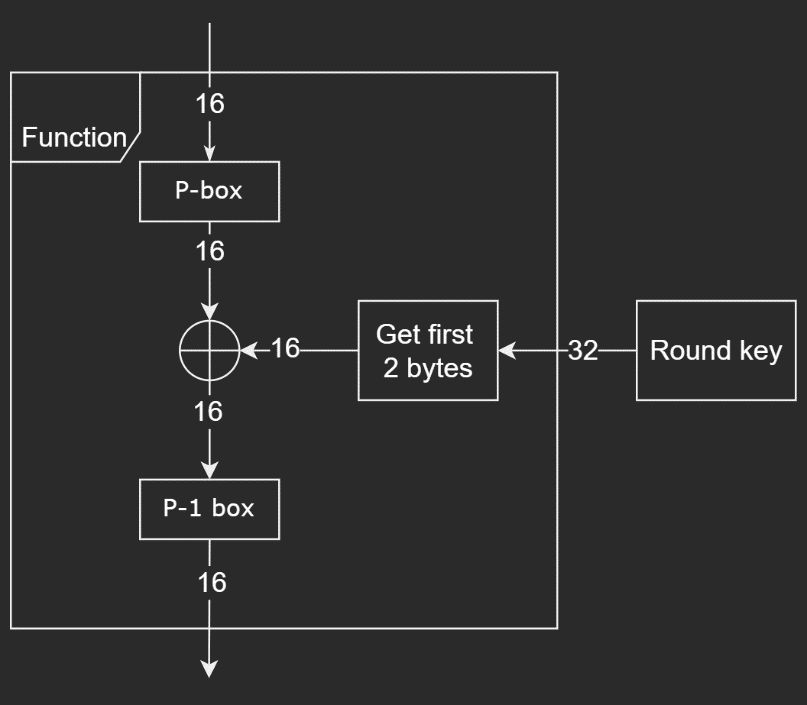
The algorithm uses 2 rounds to encrypt both blocks of the input.

The round key is dependent on the previous round key.

And is hashed to increase ciphertext complexity.

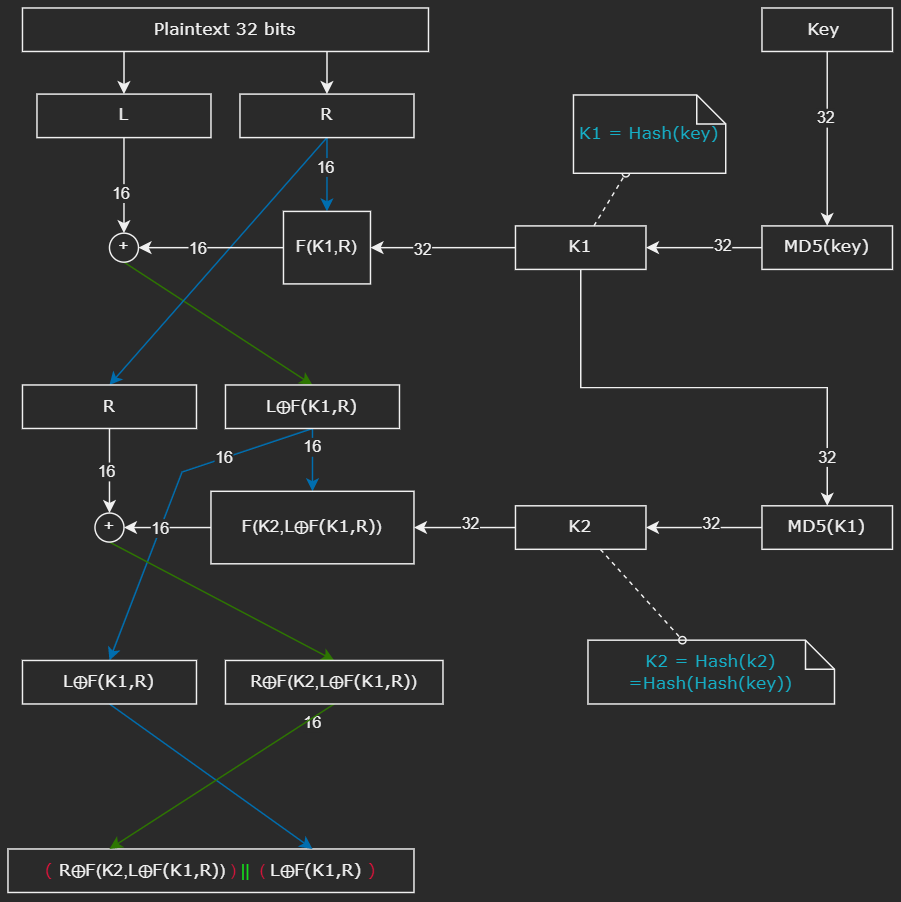
Finally, the ciphertext is swapped.

**Structure diagram of the FUNCTION:**

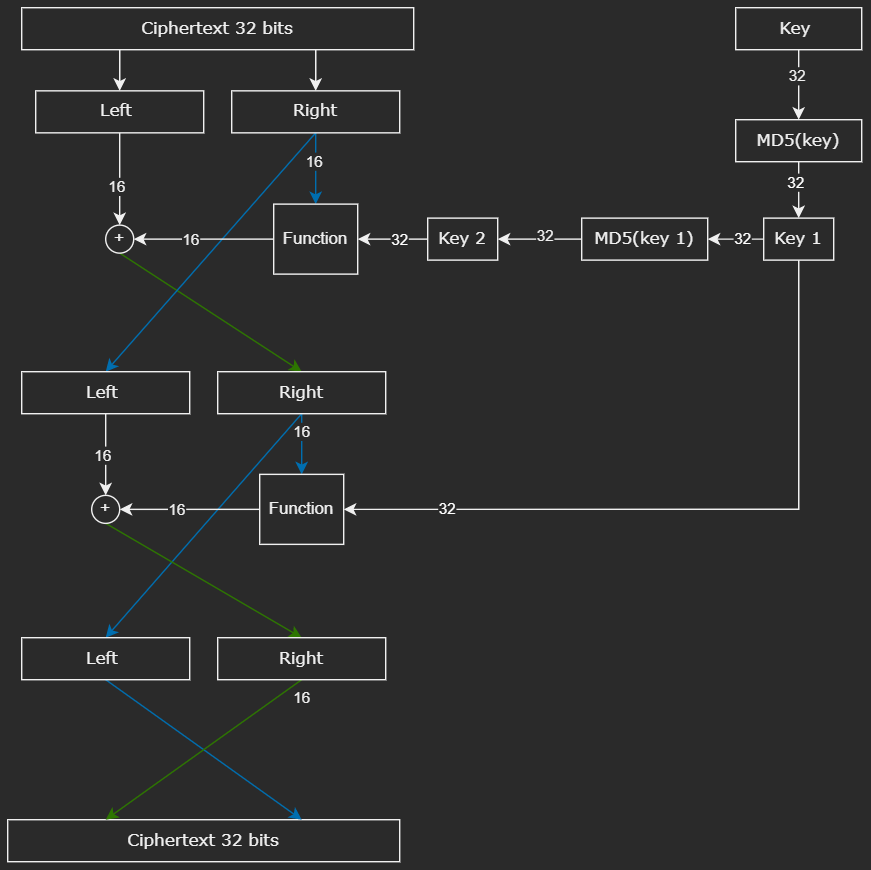


There are two Exclusive OR (XOR) operations, one inside the function and the other is applied after the function with “Left” block of input.

**Logical view of Encrypt.java process:**

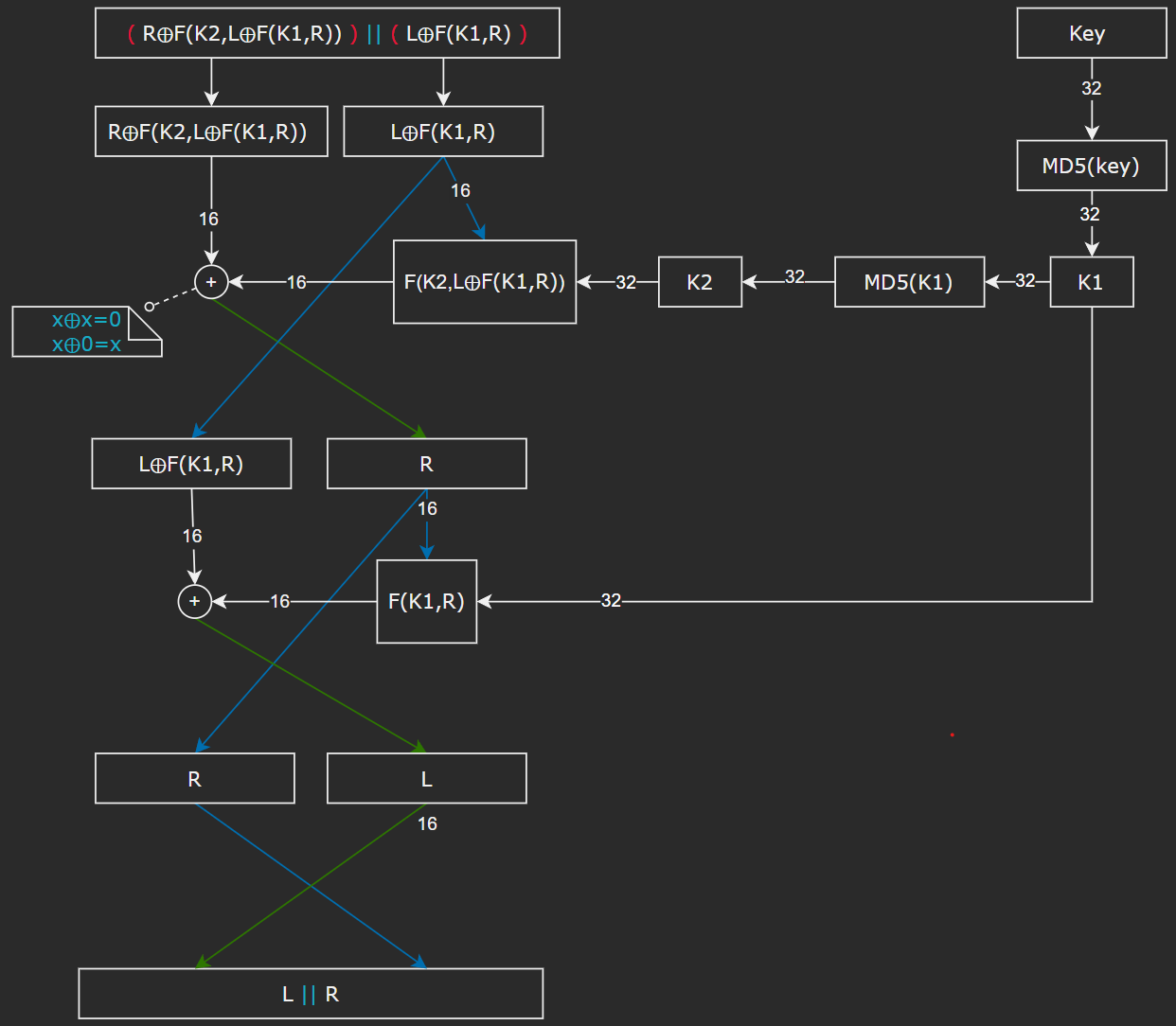


**Decrypt.java structure:**



The reverse copy of encrypt.java operations, a direct implementation of Feistel system where k1, k2 in encryption are swapped by k2, k1 in decryption.

**Logical view of decrypt.java process:**



**Code Deployment:**

First compile the source codes (javac <filename.java >)

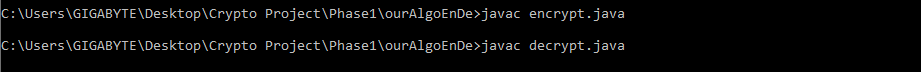
To encrypt a file, you need to specify its extension as follows:

**java ./encrypt.java <key hex> <inputFile.extenstion> <outputFile.extension>**

To decrypt a file, you need to specify its extension as follows:

**java ./decrypt.java <key hex> < inputFile.extenstion> < outputFile.extension>**

**example:**



Text

Description automatically generated



**Known Weaknesses:**

Since MD5 is used as a mask for the key, the attacker needs to find a hash collision to decrypt the ciphertext. However, if the file has a signature, or the file format is known, it will be easy to brute force all keys 2^32 with the known signature.