**DynBlock Overview**

**DynBlock** is an encryption scheme that takes plaintext and a secret key as inputs, processes them through various stages of transformation, and outputs a ciphertext. The encryption process involves multiple stages such as key expansion, block segmentation, permutation, substitution, cyclic shifts, and the use of a Toffoli gate. This MATLAB code outlines a detailed encryption/decryption process of DynBlock, where the secret key undergoes various transformations, including block division, key expansion, XOR operations, dynamic substitution, permutation, and a Toffoli gate-based operation. These stages introduce security measures like confusion and diffusion, making it more resistant to cryptanalysis.

**Data Generation**

The data in the code is generated through a series of transformations and manipulations, mainly involving the plaintext, secret key, and various cryptographic operations. Here’s a breakdown of how the data is generated throughout the encryption process of DynBlock:

**1. User Input:**

* **Plaintext:** The user is prompted to enter the plaintext (the original message) they wish to encrypt. The plaintext is then converted into binary form.
* **Secret Key:** The user is also prompted to enter a 64-bit secret key, which is essential for encryption and decryption.

**2. Input Stage**

* **Plaintext Input:** The user is prompted to enter plaintext, which will be encrypted.
* **Secret Key Input:** The user is prompted to enter a secret key of exactly 64 bits, which will be used throughout the encryption process.

**Encryption**

The main phases of encryption are as follows:

**1. Secret Key Conversion (convertToBinary)**

This function checks if the secret key is already in binary format. If not, it converts the key to binary by first converting each character to its ASCII value and then to binary. It ensures that the binary key is exactly 64 bits, either by truncating or repeating the key.

**2. Block Size Calculation (blocksizecalc)**

This function calculates parameters a, b, and c using a series of bitwise operations on the secret key. It divides the key into four parts, performs 2’s complement operations, swaps and XORs these parts, and calculates the block size based on these values.

**3. Key Expansion (keyExp)**

The keyExp function modifies the secret key based on the calculated block size. The secret key is appended with different combinations of parameters a, b, and c depending on the block size, thus generating a new expanded key.

**4. Block Segmentation (segment\_plaintext)**

This function converts the plaintext into a binary format and segments it into blocks based on the calculated block size. If the plaintext's length is not a multiple of the block size, it pads the plaintext with bits derived from the secret key.

**5. Shift Value Calculation (shiftvalue)**

A shift value is calculated based on the binary form of the plaintext. It uses the mean of the decimal values of the plaintext's binary representation to calculate a shift value, which influences the cyclic shift in later stages.

**6. Key Scheduler (key\_scheduler)**

The key\_scheduler function generates a sequence of keys by applying an XOR operation between the expanded secret key (k1) and a newly generated key (k0). The function ensures that the keys match the length of the blocks and iteratively generates keys for all blocks in the plaintext.

**7. Dynamic Permutation (dyn\_perm)**

In this step, the blocks are permuted dynamically using a left bitwise rotation. The amount of rotation is determined by the value derived from the secret key (a) and its first 8 bits, which are used in the permutation process.

**8. Dynamic Substitution (dyn\_sub)**

Dynamic substitution involves XORing the permuted blocks with the concatenated parameters a, b, and c to create new substituted blocks. This adds a layer of obfuscation to the encrypted message.

**9. Cyclic Shift (cyclic\_shift)**

This function performs a cyclic shift on the substituted blocks by the calculated shift value. The shift is done in a circular manner, ensuring that the shifted values wrap around when they go past the boundaries.

**10. Second Round of Dynamic Substitution (dyn\_sub)**

The second round of dynamic substitution works similarly to the first, applying XOR with the concatenated a, b, c values to further modify the blocks.

**11. Toffoli Gate Implementation (customToffoli2)**

The Toffoli gate is a reversible logic gate used here to introduce additional non-linearity into the encryption process. It operates by performing a logical AND between two key values, followed by an XOR with the block data.

**12. Ciphertext Generation (cipher\_text)**

The final ciphertext is generated by XORing the shifted rows of the substituted blocks with the keys. The XORed result is then concatenated to form the final ciphertext, which is displayed as the encrypted message.

**Decryption**

The main phases of decryption are as follows:

This code implements block cipher with several stages like key expansion, substitution, permutation, and XOR operations. It involves decryption processes, and the code utilizes a custom Toffoli gate, which is a reversible logical gate, for its operations. Here’s an explanation of the major sections of the code:

**1. Ciphertext and Key Input**

The user is prompted to input a ciphertext and a 64-bit secret key for decryption. The ciphertext is stored in cipher, and the secret key is captured in secret\_key. The secret key is then converted into binary form using the convertToBinary function to ensure it is exactly 64 bits.

**2. Block Size Calculation**

The function blocksizecalc is responsible for dividing the secret key into 4 parts (16 bits each) and performing XOR operations and two's complement to generate parameters (a, b, c) that will affect the block size. This helps in determining the block size for subsequent operations, which is essential in determining the granularity of the encryption process.

**3. Key Expansion**

The keyExp function generates a modified secret key based on the block size. Depending on the calculated block size, the secret key is concatenated with portions of the values a, b, and c from the block size calculation step. This key expansion process is essential for ensuring the key has enough entropy for the cipher.

**4. Block Division**

In the dec\_blocks function, the ciphertext is divided into blocks of the calculated block size. This is important for processing each segment of the ciphertext individually during the encryption or decryption process.

**5. Key Scheduling**

The key\_scheduler function generates a sequence of keys (k1, k2, ...) for each block by applying XOR operations between previous keys. The process ensures that each block of ciphertext is decrypted using different parts of the secret key.

**6. XOR Operations**

The dec\_ciphertext function applies an XOR operation between the blocks of ciphertext and the corresponding keys generated from the key scheduler. XOR is a fundamental operation in many cryptographic algorithms due to its reversible and simple nature.

**7. Toffoli Gate**

The custom Toffoli gate implementation (customToffoli2) is used for creating controlled operations. This reversible logic gate performs a conditional XOR operation based on the input values, which is useful in many quantum computing and cryptographic contexts.

**8. Dynamic Substitution and Permutation**

The dec\_substitution function performs dynamic substitution by XOR-ing the data with concatenated parameters (a, b, c) to introduce confusion in the ciphertext. The dec\_cyclic function shifts the bits cyclically, creating diffusion in the ciphertext to make the encryption harder to analyze. Afterward, the dec\_perm function applies a permutation based on a rotation using values derived from the secret key and the parameter a.

**9. Final Plaintext Generation**

The decplain\_text function combines all the processed blocks, converts them back into bits, and extracts the original message (without any padding added during encryption). This function ensures that the final output is the original plaintext that was encrypted with the cipher.