FrankenCipher: A New Simple Block Cipher

Gerardus Samudra Sembiring[1], **Nathaniel Evan Gunawan**[2].

[1,2] Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika (STEI), Institut Teknologi Bandung (ITB), Jalan Ganesha 10, Bandung 40132
E-mail: 13516103@std.stei.itb.ac.id, 13516055@std.stei.itb.ac.id

Abstract. Encryption, the process to obfuscate information deemed confidential or private, while dating back to the days of Ancient Greece and Rome, has now rapidly risen in complexity and prominence, thanks to the development of modern technology. We will now take a look at several modern block cipher encryption algorithms, and propose our own block cipher encryption algorithm, applying in it the Feistel structure, confusion and diffusion, substitution and transposition, and iterated ciphers. **Keywords**: encryption, block cipher, Feistel, confusion, diffusion.

1. Introduction

In this paper, we would like to propose a new, simple block cipher titled FrankenCipher. FrankenCipher arises out of the need for a block cipher that is more performant, yet maintains the level of security that historically well-established block ciphers such as DES offer. To achieve this, we combine the unbalanced Feistel network implemented in MacGuffin, a 64-bit block cipher, and the key scheduling algorithm of RC4 for the S-block, and the structure of Twofish.

2. Theoretical Framework

2.1. Block ciphers

A block cipher is an encryption technique that operates on the block level. To encrypt a plain text with a block cipher, the plaintext is split into multiple blocks of *n*-bits. The block cipher then proceeds to encrypt each block to produce a cipher block, utilising a fixed, symmetric key; that is, the same key is used for both the encryption and decryption process. Finally, the cipher blocks are concatenated together to form a ciphertext.

2.2. Modes of operation

There are a number of modes of operation for block ciphers, 3 of which are relevant to the topic of this paper and shall be briefly explained:

- Electronic code book (ECB)
  - ECB works by independently encrypting the message blocks, passing them into an encryption function with a key that is shared by all of the blocks. (Hence, 2 or more blocks containing the same message produce the same number of identical cipher blocks.) This results in a mode that is fault-tolerant, since all blocks are encrypted independently of each other, but vulnerable to statistical attacks.
- Cipher block chaining (CBC)

    - CBC introduces a dependency between the resulting cipher blocks; a cipher block, in this mode, depends on its corresponding plain text as well as all the cipher blocks produced before it. The first message block is XOR-ed with a random initialisation vector (IV) to ensure that each message is unique, and then processed by an encryption algorithm with a key, resulting in a cipher block. The second message block will be XOR-ed with the first cipher block, the third message block with the second cipher block, and so on. In this way, CBC is harder to cryptanalyse than ECB, but a corrupted message block will corrupt all the cipher blocks after it, due to the chaining between the blocks.
- Counter mode
    - Counter mode attempts to improve upon CBC by removing the chaining, and therefore dependency, between the cipher blocks. This mode utilises a counter, a block with the same size as the message cipher, which is initialised for the first block encryption and incremented for every subsequent block. For each message block, the counter and key will be fed into the encryption algorithm, then XOR-ed with the message block to form a cipher block.

2.3. Balanced Feistel networks

A Feistel cipher, or more commonly known as a Feistel network, is a symmetric structure used by a number of prominent block ciphers, including the DES, GOST, Blowfish and Twofish ciphers. For each message block, it works by splitting the block into two equal pieces: L_0 and R_0. For each round $i = 0, 1, 2, ..., n$, we compute:

$L_{i+1} = R_i$

$R_{i+1} = L_i \oplus F(R_i, K_i)$

The resulting ciphertext is (R_{n+1}, L_{n+1})

This ciphertext can be decrypted by using the same process, for $i = n, n-1, n-2, ..., 0$, resulting in (L_0, R_0).

2.4. Unbalanced Feistel networks

It is possible to define a Feistel network that has unequal left and right halves. This is explored in Schneier and Kelsey 1996, defining an *Unbalanced Feistel Network (UFN)*. In a UFN, the n bits of input are split into two pieces of s and t bits each. The s-bit piece is the *source*, being the s most-significant bits of the input, and the t least-significant bits of the input make up the *target*.

For each round, the source block is fed into the round function F along with the round key, producing a t-bit output, which is XOR-ed with the t-bit *target*. Now the t-bit result forms the most significant bits of the next round's input, while the s-bit source block is passed unmodified into the least significant bits.

One round of a UFN is therefore:

$X_{i+1} = (F(msb_s(X_i), k_i) \oplus lsb_t(X_i)) || msb_s(X_i)$

3. Proposed block cipher

3.1. Definitions

The FrankenCypher uses a block size of $n = 96$ bits and a variable-length key K of up to 2048 bits. We use an unbalanced Feistel network with a split of $s = 64$ and $t = 32$ bits, iterated for $N = 16$ rounds. The round function F takes in the s-bit input along with a 64-bit round key k_i. In every round, F uses four bijective 8-by-8-bit S-boxes $S_0 ... S_3$, which are key-dependent.

3.2. S-box initialisation

First, take an initial S-box, S_{-1}, consisting of the identity permutation.

We first encrypt the main key K under four constant keys $C_0 ... C_3$, using the main encryption algorithm (defined later) in ECB mode, in order to generate four derived keys $DK_0 ... DK_3$. In this operation, $S_0 ... S_3$ are not yet available, se S_{-1} is used. The derived keys are then used with the RC4 key scheduling algorithm to generate the S-boxes $S_0 ... S_3$.

$C_0 = 0x4672616e6b53626f784e756c$
$C_1 = 0x4672616e6b53626f784f6e65$
$C_2 = 0x4672616e6b53626f7854776f$
$C_3 = 0x4672616e6b53626f78547269$

3.3. *Round key generation*

The round keys are generated

3.4. *Round function*

The round function takes in a 64-bit input and a 64-bit round key, and produces a 32-bit result. First, the input bits are split into two; 32 even-numbered bits 0..62 into w_1 and 32 odd-numbered bits into w_2. Each word is then passed into a function g parameterized by the two 32-bit halves of k_i to produce two 32-bit results, which are then added modulo 2^{32}. The bits of this sum are reversed in order, creating the output.

Function g mixes the input word with the round key word using a mod-2^{32} addition, and then splits the sum word into 4 bytes. The four bytes are substituted by the four S-boxes, resulting in 4 bytes. The 4 bytes are output back into F.

4. Figures

5. Tables

6. Conclusion and future development

7. References

[1] Blaze M and Schneier B, 1995, *The MacGuffin Block Cipher Algorithm*

[2] Schneier B, Kelsey J, Whiting D, Wagner D, Hall C, Ferguson N, 1998, *Twofish: A 128-Bit Block Cipher*

[3] Schneier B and Kelsey J, 1973, Unbalanced Feistel Networks and Block-Cipher Design

Acknowledgments

Authors wishing to acknowledge instruction from the lecturer, as well as assistance or encouragement from colleagues.