FareCipher: Block Cipher Algorithm

An ARX-based Block Cipher

Fairuz Astra Pratama & Resa Kemal Saharso
Computer Science & Informatics
Institut Teknologi Bandung
Bandung, Indonesia
13514104@std.stei.itb.ac.id, 13514109@std.stei.itb.ac.id

***Abstract*—Fare Cipher is a block cipher algorithm that is based upon the commonly used ARX operation (involving modular addition, bitwise rotation, and XOR operation) and Feistel Architecture. The main difference of this cipher is that it operates on several unit of works, combining the use of ARX operation on 8-bit, 32-bit and 64-bit words to encipher the message. This cipher works by first expanding the encryption key into several subkeys that is then used in each enciphering rounds as a parameter to the Feistel's round function.**

Keywords—block cipher, ARX operation, Feistel architecture

I. INTRODUCTION

In this paper, we will introduce a new block cipher algorithm called FareCipher, which consist of the approach used when designing it, the justification for using said approaches, the algorithm specification, as well as security and performance analysis of this cipher. Before getting to the algorithm specification, we will first review the block cipher design principles that is used when designing this cipher, and also review the AES block cipher as an example of a commonly used block cipher.

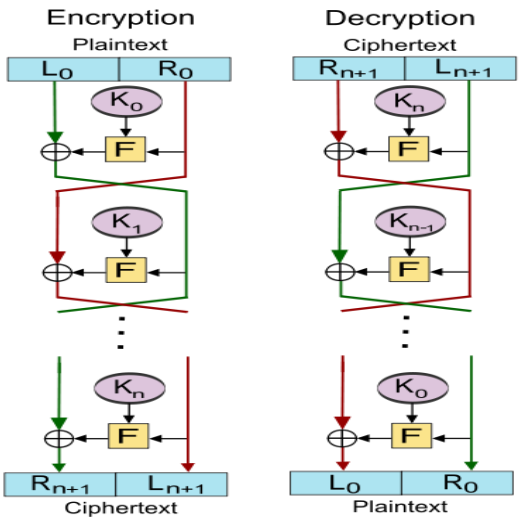
A. Block Cipher Review

Block cipher is a category of encryption algorithm that performs encryption and decryption on a fixed-sized chunk called block. As opposed to stream cipher which operates on each bit of the message by XOR-ing it with a keystream, block cipher take a block of the plaintext, convert it into the corresponding ciphertext block, and vise versa. When designing a block cipher, there are a couple of design principles and/or methods that should be taken into account:

1. **Shannon's Principle of Confusion and Diffusion**
   In order to be resilient against statistical attack, a block cipher had to fulfill these principles:
   a. Confusion principle states that the connection between plaintext, key, and ciphertext should be made as complex and involved as possible. This can be realized by using a complex substitution process, such as S-Box or modular addition.
   b. Diffusion principle states that each bit of the ciphertext should depend on many bits of the plaintext and key. The general idea is that a change in one of the ciphertext/key bit should resulted in about half changed bit in the ciphertext.

2. **Iterated Cipher**
   An encryption algorithm can be made of an iteration of relatively simple transformation. Each transformation iteration will transform it's input using a subkey, where each iteration will have a different subkey. The subkey is usually derived and/or expanded from the encryption key.

3. **Feistel Network**
   Feistel network is a commonly used version of an iterated cipher architecture. In this architecture, the input block is divided into two equal size. Each round, one half of the block is transformed by the round function "*f*" (using the iteration subkey) and is XOR-ed by the other half. After that, both of the half is swapped before the next iteration. This network is reversible, which means that the round function "*f*" can be used for both encryption and decryption, and doesn't need to be invertible.



4. **S-Box**
   S-Box is a substitution method that is basically a lookup table, mapping several input bits to several output bits. This method is usually the only source of nonlinearity in

an encryption algorithm, and are used in commonly used block cipher such as AES.

B. AES Review

Also known as Rijndael, AES is one of the most used block cipher today. AES is a block cipher that uses a 128, 192, or 256 bit key, and have a block size of 32 byte. This cipher consist of 10, 12, or 14 rounds of enciphering (according to the key length). Each round will consist of several transformation, and will use a different ubkey that is derived/expanded form the encryption key.

In AES, the input and output state is represented in a matrix of byte. This means that the 16 input block and all of the round's input are output are represented in a 4x4 byte matrix. The subkey length is the same as the block size, and is also represented in 4x4 byte matrix. Below are a general explanation of how the four transformation that is used in each round works:

1. **SubBytes**: Input bytes substitution using a constant S-Box
2. **ShiftRows**: Circular shift of each of the the input's rows.
3. **MixColumns**: Garble the input columns via multiplication with a constant matrix.
4. **AddRoundKey**: XOR between the input and the round key.

Since each round will need a 16 byte subkey, the encryption key will need to be expanded into several subkeys. AES does this by a chain of circular rotation, S-Box substitution, and XOR operation. After the key is expanded into several subkey, the next step is the iterative enciphering itself. In general, the transformation that happened in each enciphering round are as such:

1. For the first enciphering round,
   a. **AddRoundKey**, XOR the input (initial plaintext) with the round key
2. For the rest of the enciphering round,
   a. **SubBytes**
   b. **ShiftRows**
   c. **MixColumns**
   d. **AddRoundKey**, XOR the input (output of the previous round) with the round key
3. After the final enciphering round,
   a. **SubBytes**
   b. **ShiftRows**
   c. **MixColumns**

II. FareCipher Specification

FareCipher is a block cipher that uses a **256 bit key** and have a **block size of 32 byte**. This cipher consist of **32 rounds of enciphering** done using the **Feistel network** architecture. On each iteration, the round function "f" will receive half of the 32 byte block (16 byte of data) and output a 16 byte of garbled data using a **128 bit subkey** that is expanded from the 256 bit key. This cipher consists of two main function: the **subkey generation**, as well as the **round function** that will be used in each rounds of enciphering. The round function will be assembled into two Feistel network: one that is used for encrypting message, and the other that is used to decrypt the message. On both network, the same subkey generator will be used to generate the subkeys used each round function in the Feistel network.

A. Approach Used

In most block cipher, encryption is done in a number of rounds, where each round map the previous round output into the next intermediary output. Each round also need a key parameter, which is usually derived / generated form the encryption key. For example, if an encryption is done in 20 round, each of which require 128 bit subkey, the encryption key will be expanded into 20x128 bit subkey. The general idea of FareCipher is to make the Subkey generation do most of the heavy-lifting, generating a subkey for each of the rounds from the encryption key (almost like a short keystream generator). The subkey generator will focus on fulfilling Shannon's principle of confusion, making each subkey bit depends on several encryption key bit. This way, subkeys generated by different encryption keys will vastly differ.

Using this method, the round function used in each iteration does not need to care about fulfilling Shannon's principle of confusion, since simply XOR-ing the plaintext with the subkey will guarantee that the relationship between key, ciphertext, and plaintext remains complex. This way, the process if encrypting / decrypting each of the block in a message can be processed more quickly. This is because while the round function will be called a number of times for each block that needs to be encrypted or decrypted, the subkey generator only needed to be called once for each encryption key used.

FareCipher uses three simple operation during the encrypting and/or the decrypting process, which is modular addition, bitwise rotation, and bitwise xor. This operation also usually known as ARX operation and is usually used in modern stream cipher such as XChacha20. Modular addition and xor is used as a substitution process, while the bitwise rotation is used as a transposition method. Modular addition is also used as a source of nonlinearity, which can also be provided by using S-Box lookup. The main difference between the usual ARX operation, is that while most cipher operates on a fixed unit of work such as a 32 bit word, FareCipher operates on several unit of works, combining the use of ARX operation on 8-bit, 32-bit and 64-bit words to encipher the message.

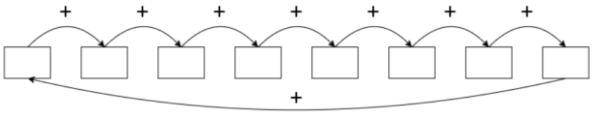
Although S-Box can garble data much faster and thoroughly than a chain of modular addition, it has several weakness. One of which, is that in most platforms S-Box lookup are not executed in constant time, since the array lookup processing time depends on the index that is being searched. This resulted in an attack known as cache-timing attack which has been proved successful against OpenSSL implementation of AES. Since constant-time S-Box are generally too slow for general

usage, the use of modular addition as a source of nonlinearity can solve this problem, since it is processed in constant time.

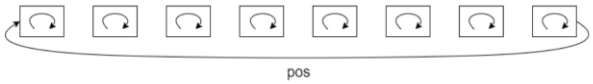
B. Subkey Generations

The subkey generation will expand the 256 bit key into 32 128 bit subkey. This operation is based on ARX operation and will operate on 32 bit unsigned integer. This means that the 256 bit key will be interpreted as an array of eight 32 bit unsigned integer and each of the 128 bit subkey will be interpreted as four 32 bit unsigned integer. The operation used in the subkey generation is a combination of:

1. **Modular addition between the encryption key integers**. This operation is equal to "a + b mod 2^32". During this operation, every integer in the key integer array will add its value to the next element and element in the last index will add its value to element in the first index. This is done to garble the key value itself, so that it differs for each iteration.

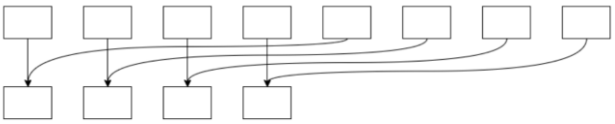


2. **Bitwise rotation of the encryption key integer as well as array element rotation**. During this operation, each key integer will be circular-shifted to the left by one bit. After that, the integer array itself will be circular-shifted to the left by one element. This is done to garble the key value.



3. **XOR operation between the encryption key integer to make the subkey**. In this operation, eight key integer will be combined to make four subkey integer using XOR operation. This is done using the formula:

$$subkey[n] = key[n] \wedge key[n+4], 0 \leq n \leq 3$$

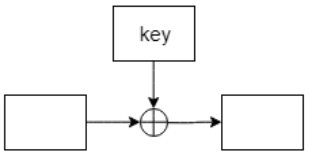


The subkey is generated by first repeating the first two step 16 times to garble the encryption key content. This step is done so that similar key will still produce a vastly differing subkeys. After that, by repeating all three operations above 32 times, where each iteration will generate a subkey for the corresponding round, the 256 bit key will be expanded into 32 128 bit subkey. Each subkey will then be used as a parameter to the round function in the corresponding encryption iteration/round.

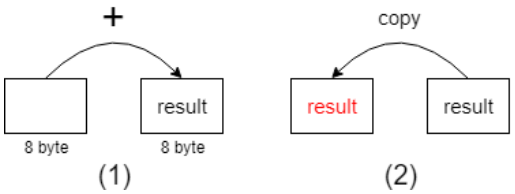
C. Round Function

The round function transforms 16 bytes of input into 16 bytes of output which means in each iteration of the round function half of the block is processed. It consists of 3 operations:

1. **XOR operation between the block and the subkey**. In this operation, 16 bytes of plaintext will be combined with 16 bytes of subkey to make 16 bytes of ciphertext using XOR operation.



2. **Modular addition of the first and second half of the block and duplication of the result**. Similar to the modular addition in the subkey generation, 16 bytes of ciphertext is split into 2 8 byte integers and it's values are added creating a single 8 byte value. The resulting value is then duplicated to restore the 16 byte length of the block.



3. **Bitwise rotation of the block**. During this operation, the block will be circular-shifted to the right by one bit.

The function is simple because it is only for fulfilling Shannon's diffusion principle while the confusion principle has been handled by the subkey generation. The multiple rounds of Feistel network will also enhance the diffusion effect. Since the key generation has to be called only once for an encryption key, every block that is encrypted using the same key can be encrypted in a relatively short period of time.

D. Feistel Network

The feistel network that is used for encrypting and decrypting message is the standard Feistel network configuration. For encryption, the right-half of the block will be transformed using the round function above, and xor-ed by the left-half before being swapped. For decryption, the opposite is true, the left-half will be transformed. The subkey generator will be used in the same way for both encryption and decryption, but the order of which the sub key is being used is different. For encryption, the first round function iteration will use the first subkey, and incrementing the subkey index for each iteration. For decryption, the first round function will use the last subkey, and decrementing it's index for each iteration.

III. Simulation and Analysis

In this paper, we implement the FareCipher specified above in C programming language in order to simulate the encryption process as well as analyze its security properties. Byte arrays are used for storing key and plaintext, generating subkeys and modifying blocks based on the round function. 2 operation modes are implemented for this cipher, which are the Electronic Code Book (ECB) and the Cipher Block Chaining (CBC). Electronic Code Book operation mode encrypts and decrypts blocks individually, while Cipher Block Chaining operation mode makes it that the processing of a block depends on the result of the previous block thus making a dependency between blocks. Below are examples of a plaintext and the resulting ciphertext:

Key:
27 66 72 73 100 101 104 110 119 120 122 129 132 135 138 142 144 151 159 160 196 212 214 220 224 234 235 237 238 241 248 252
Plaintext:
3 11 28 30 36 42 43 51 61 63 72 88 96 98 111 115 118 120 123 164 165 166 174 177 179 198 197 200 242 248 249 250
Ciphertext:
11 103 172 156 37 204 50 2 157 14 103 232 213 248 174 43 170 194 2 104 171 230 60 72 123 34 118 255 33 22 98 209

The shown example is calculated in 5482 nanoseconds for the key expansion and 10.222 nanoseconds for block encryption. The git repository containing the code can be accessed in https://github.com/FairuzAP/FareCipher.

A. Security Analysis

1) Plaintext Difference

Based on the example above, it can be seen from comparing the plaintext and ciphertext that the cipher has fulfilled Shannon's confusion principle. In addition, based on another example where the plaintext is changed by 1 bit (highlighted):

Plaintext : 0x30B1C1E242A2B323D3F485860626F7376787BA4A5A6AEB1B3C6C5C8F2F8F9FA
Ciphertext : 0xB44791DDA886A0C9D2DB2692ABCF624AAC4E22F7AD5DF3B7B2496B8F02581A2

The plaintext and ciphertext has a difference of 119 bits which means almost every bit of the ciphertext is different from the plaintext, thus also achieving the diffusion principle. This also guarantees that a bit difference of more than one will also change the ciphertext drastically.

2) Ciphertext Difference

A 1 bit difference in ciphertext (highlighted) with the same key results in a vastly different plaintext:

Ciphertext : 0xB6 0x7A 0xB9 0xC2 0x5C 0xC3 0x20 0x29 0xD0 0xE6 0x7E 0x8D 0x5F 0x8A 0xE2 0xBA 0xAC 0x20 0x26 0x8A 0xBE 0x63 0xC4 0x87 0xB2 0x27 0x6F 0xF2 0x11 0x66 0x2D 0x1
Plaintext result : 0x31 0x04 0xE2 0x87 0x65 0x51 0x4F 0x53 0xD2 0x41 0xD6 0xE3 0x21 0xD5 0x0B 0x57 0x65 0x04 0xEF 0x61 0xEC 0x59 0x50 0xFB 0x3E 0xEF 0x09 0xA4 0x99 0xBC 0x24 0x4

This means that there are no correlation in similar ciphertexts thus preventing cryptanalysis on the ciphertext.

3) Key Difference

A 1 bit difference in key (highlighted) with the same plaintext also results in a vastly different ciphertext:

Key : 0x1B4248496465696E77787A8184878A8E90979FA0C4D4D6DCE0EAEBEDEEF1F8FC
Plaintext : 0x30B1C1E242A2B323D3F485860626F7376787BA4A5A6AEB1B3C6C5C8F2F8F9FA
Ciphertext : 0x3E430D893D10DB08F3CE13FDBF806D3E73DE469C0E80AFB202D3EF71DAB435A8

4) Key Size

The FareCipher algorithm has a key size of 256 bits, thus having 2^{256} possible combination of keys. This large number of possiblities will slow down attackers that tries to brute force the algorithm by using every single key possible.

5) Calculation Time

Comparing it's calculation time to the AES cipher (7382 nanoseconds for key expansion and 1470 nanoseconds for block encryption) shows that the key generation is actually faster, albeit it's block encryption being very slow. It is predicted that the slow block encryption is caused by the nonoptimal implementation of the cipher.

IV. Conclusion

FareCipher has been proven to be working as a block cipher algorithm, and messages can be successfully encoded and decoded using this method. We also analyzed the diffusion of this cipher and has proven that small changes in the key and plaintext resulted in largely differing ciphertext. For further research we recommend doing a more indepth cryptanalysis of the cipher, as well as making a more efficient implementation of the cipher to compare its performance against other similar block cipher.

References

[1] https://pdfs.semanticscholar.org/307d/c553e897981853845b6c846e81ba0f80ee99.pdf

[2] https://cr.yp.to/antiforgery/cachetiming-20050414.pdf
[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2017-2018/Advanced-Encryption-Standard-(AES)-(2018).pdf