THE CAST-256 Encryption Algorithm

Report by Noeler Kappa (181509)

Table of Contents

Abstract

The CAST_256 also known as CAST 6 is a symmetric-key, block cipher that was submitted as a candidate for the Advanced Encryption Standard. It has a variable key size of 128, 160, 192, 224 or 256 bits. It is an extension of CAST-128 both of them were designed according to the 'CAST' design methodology. This design is based on the Generalised- Feistel- Network.

Introduction

As a symmetric key algorithm, it uses the same cryptographic keys for both encryption of plaintext and decryption of ciphertext. The full CAST-256 (for encryption) consists of 48 rounds for all key sizes: 6 forward quad-rounds + 6 inverse quad-rounds. There are likewise, (for decryption) 48 rounds, and the scheme is exactly the same: 6 forward quad-rounds + 6 inverse quad-rounds, because an inverse quad-round is just a quad-round upside-down. Thus, the same scheme.

It is a 128-bit iterated block cipher. The algorithm encrypts and decrypts on fixed-length groups of bits called blocks. It transforms fixed-size blocks of plain-text into identical size blocks of cipher text, via the repeated application of an invertible transformation known as the round function, with each iteration referred to as a round.

Transformation by the cipher is done in two parts, the key schedule algorithm and encryption function.The key schedule algorithm is used to initialise the permutation, deriving a set of sub keys from an initial key.

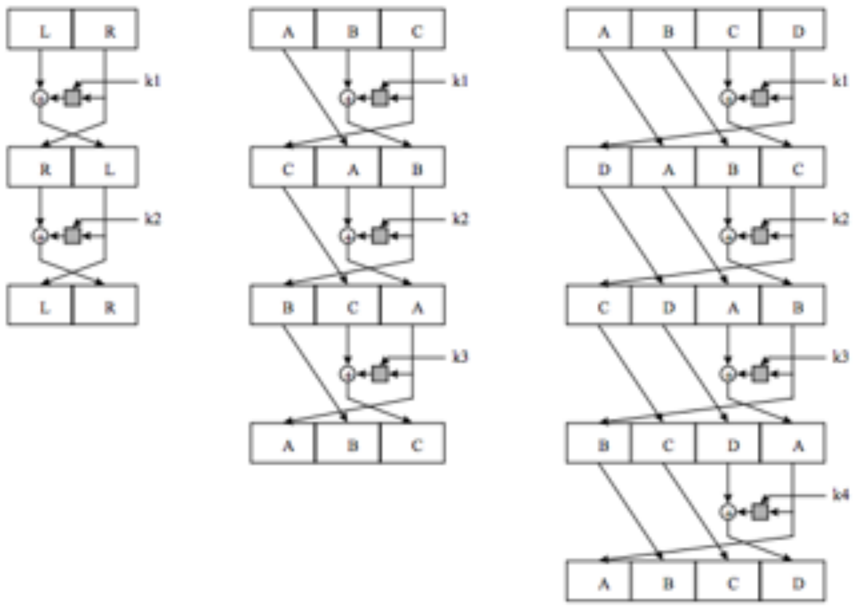
Encryption of the plain text proceeds through quad-rounds. Usually, the round function takes different round keys Ki as second input, which are derived from the original key. Key whitening which is the combining of data with portions of the key is used in addition to this. At the beginning and the end, the data is modified with key material (often with XOR, but simple arithmetic operations like adding and subtracting are also used) as seen below. Here we add round key to the intermediate vector (XOR), the iterated ciphers then performs a non-linear substitution operation/S-box on disjoint parts of the input and then an affine transformation of the whole intermediate vector.

Cast 256 is an extension of CAST-128 . As referred to in the notation for the CAST-256 encryption algorithm . Cast-128 uses fixed 8x32-bit S-boxes: for encryption and decryption (S_1, S_2, S_3, S_4) with a key schedule (I_a, I_b, I_c, I_d), round operations: +, -, <<< and three round functions: f_1, f_2 and f_3.

It takes a block of the plaintext and the key as inputs, and applies several alternating "rounds" or "layers" of substitution boxes (S-boxes) and permutation boxes (P-boxes) to produce the cipher text block. The S-boxes and P-boxes transform (sub-)blocks of input bits into output bits.

During the iteration, the block cipher uses the substitution-permutation network (SPN), it takes a block of the plaintext and the key as inputs, and applies several alternating rounds consisting of a substitution stage followed by a permutation stage—to produce each block of cipher text output. The non-linear substitution stage mixes the key bits with those of the plaintext, creating Shannon's confusion.

The CAST 256 uses the incomplete Feistal Network. The left-most diagram is the traditional Feistel network. If this describes two rounds of DES, then L & R are each 32 bits in length and the cipher has a 64 bit block size. The middle diagram refers to an extended Festal network for a cipher with a 96 bit block size and the right most diagram describes the structure of a cipher with a 128 bit block size. The number of rounds will be the basic unit(n) for submitting all input bits to the round function.



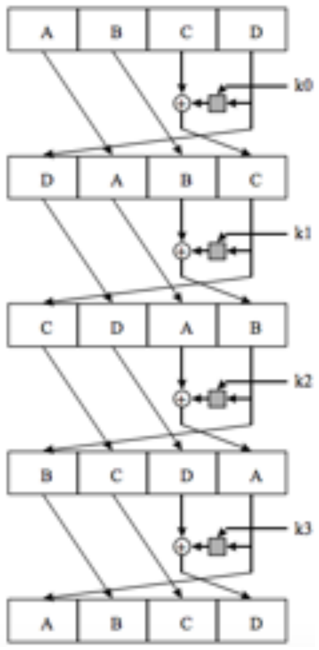
Cast 256 consist of 48 rounds or 12 quad rounds of mixing

$$\beta \leftarrow Q(\beta) \quad \begin{cases} C = C \oplus f_1(D, k_{r_0}^{(i)}, k_{m_0}^{(i)}) \\ B = B \oplus f_2(C, k_{r_1}^{(i)}, k_{m_1}^{(i)}) \\ A = A \oplus f_3(B, k_{r_2}^{(i)}, k_{m_2}^{(i)}) \\ D = D \oplus f_1(A, k_{r_3}^{(i)}, k_{m_3}^{(i)}) \end{cases}$$

"Forward Quad-Round"

$$\beta \leftarrow \overline{Q}(\beta) \quad \begin{cases} D = D \oplus f_1(A, k_{r_3}^{(i)}, k_{m_3}^{(i)}) \\ A = A \oplus f_3(B, k_{r_2}^{(i)}, k_{m_2}^{(i)}) \\ B = B \oplus f_2(C, k_{r_1}^{(i)}, k_{m_1}^{(i)}) \\ C = C \oplus f_1(D, k_{r_0}^{(i)}, k_{m_0}^{(i)}) \end{cases}$$

"Reverse Quad-Round"



Implementation

An easy way to design this is to design the structure such that if there are r rounds in the full cipher, the first r/2 rounds use right-shifting (as shown in the diagram above) and the last r/2 rounds use left-shifting. In this way, the desirable feature of “traditional” Feistel networks with respect to decryption (i.e., that decryption is identical to encryption, requiring only a reversal of the round keys) is preserved.

It uses an f-function that produces a 32-bit output from a 32-bit input, and each round consists of modifying one 32-bit quarter of the block by XORing it with the f-function of another 32-bit quarter of the block. There are 48 rounds in CAST-256. The rounds are organized in groups of four, called quad rounds. Each round uses two sub keys, a "mashing" sub key 32 bits in length, and a rotation sub key 5 bits in length

The f-function in CAST-256 comes in three different types, with the positions in which the operations of addition, XOR, and subtraction are used changed. In the first type of f-function, the input first has the current mashing key added to it. Then, the result is given a circular left shift the extent of which is given by the current rotation key. Then, each of the four bytes of the result are used to select entries from one of the four S-boxes. The entries in the S-boxes are 32 bits wide. The entry in S1, selected by the first byte, first has the S2 entry selected by the second byte XORed to it, then the S3 entry selected by the third byte subtracted from it, then the S4 entry selected by the fourth byte added to it.In the f2 function, the mashing key is XORed, the S2 entry is subtracted, the S3 entry is added, and the S4 entry is XORed. In the f3 function, the mashing key is subtracted, the S2 entry is added, the S3 entry is XORed, and the S4 entry is subtracted.

The block is divided into four 32-bit segments. First, an f1 f-function of the fourth segment is XORed to the third segment, then an f2 f-function of the third segment is XORed to the second segment, then an f3 f-function of the second segment is XORed to the first segment, and finally an f1 f-function of the first segment is XORed to the fourth segment.

An operation called an "octave" is used to generate the CAST-256 key schedule. An octave is just like a forwards quad- round, except it operates on a 256-bit block, and comprises eight rounds; first, an f1 f-function of the eighth segment is XORed to the seventh segment, and so on until an f2 f-function of the first segment is XORed to the eighth segment at the end.

CAST-256 may have a key that is 128, 160, 192, 224 or 256 bits long. The 256-bit input to the first octave consists of the key padded with zero 32-bit words in the rightwards, later positions.

The sub keys used for the octaves are as follows: the first mashing sub key is 5A827999 in hexadecimal, and the first rotation sub key is 19 in hexadecimal.

Subsequent sub keys are formed by adding 6ED9EBA1 to the previous round's mashing sub key, and adding 17 (again, hexadecimal) to the previous round's rotation sub key. Needless to say, the two additions are modulo 2^32 and 2^5, respectively.

The value of the block after each octave supplies enough sub key material for one quad round of the cipher. The eight 32-bit segments of the 256-bit block supply KR0, KM3, KR1, KM2, KR2, KM1, KR3, and KM0 in order: that is, the first segment supplies the first rotation key for the quad round (all but the last five bits are ignored), the second segment supplies the last mashing key for the quad round, and one proceeds onwards, forwards through the rotation keys and backwards through the mashing keys.

As per notation shared in the encryption algorithm, the CAST256implementation.txt file provided contains the following steps;

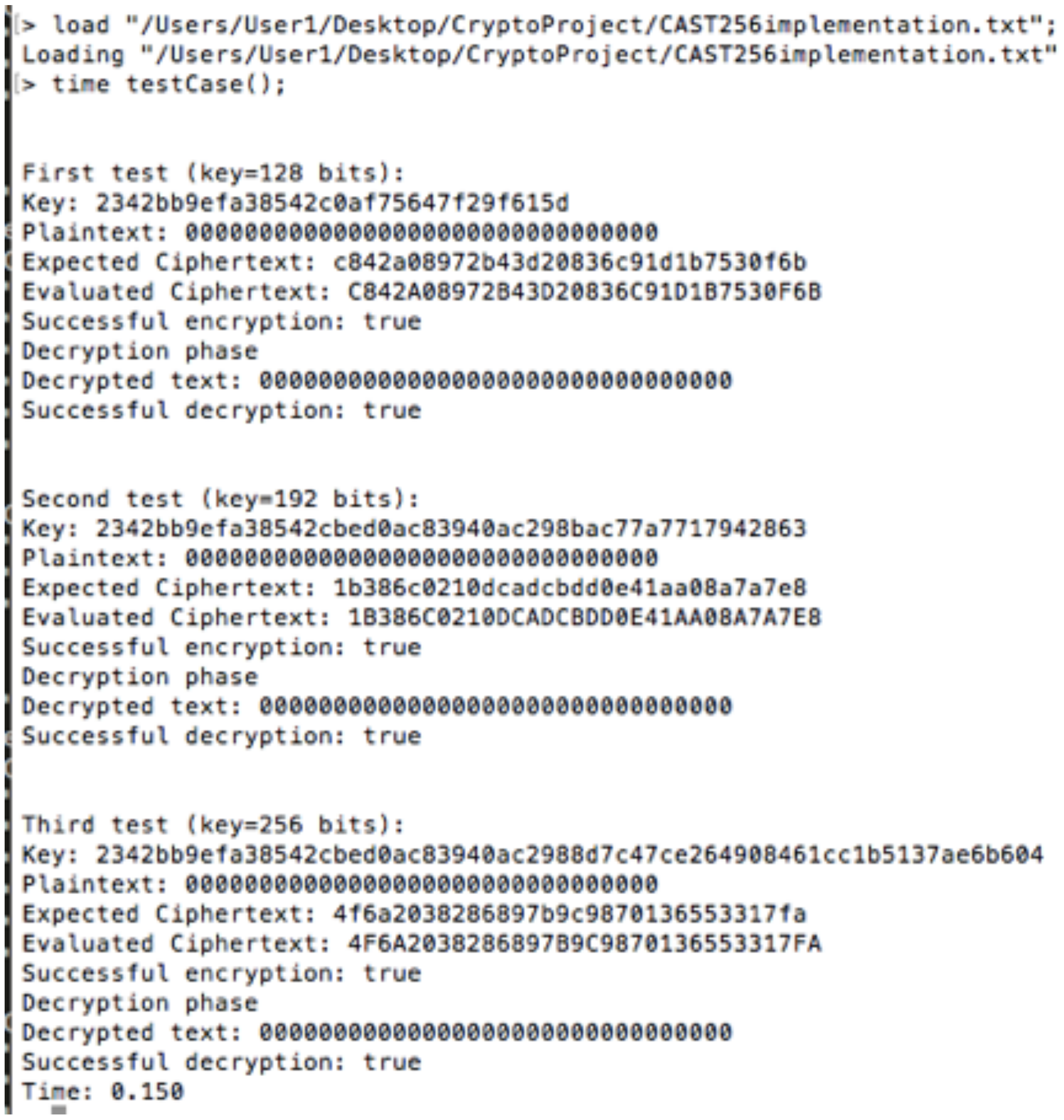
- A utility function that takes as input an integer and returns a Most Significant Bit(MSB) Sequence. The input here is an integer and the output is a MSB sequence of bits representing an integer
- For the reverse function we have the utility function that takes as input a Most Significant Bit Sequence and returns an integer. The input is a sequence and the output is an integer representation of the sequence
- For the operation of S-boxes;-
- ($SBox_1$, $SBox_2$, $SBox_3$, $SBox_4$), we take input as a byte and returns the corresponding image through the SBOX as an integer to be considered as a 32-bit sequence. The input will be a bit sequence that will be cast to an integer and used as an index. The output will be an integer taken from the SBOX
- The update functions takes an integer and update it based on a rotation key, a masking key and the SBoxes
- The next function returns the rotation and masking key. The input is the integers A,B,C,D,E,F,G,H representing the current key and the output will Kr, rotation key and Km, masking key
- The function performing the forward octave takes a, b,c,d,e,f,g,h as integers, the current round index, the masking key set and rotation key set
- The function of the key set takes as input, a key of variable length that can be 128, 192, or 256 bits and generates 4 rotation and 4 masking keys for each round. The input is a key of a hexadecimal string and the outputs are Km(list of masking keys) and Kr(list of rotation keys)
- The QBAR function performing the inverse quad round takes integers A,B,C,D, Kr the shift keys and Km the mixing keys and the outputs are the A,B,C,D (integers)
- The Q function performs a forward quad round takes integers A,B,C,D, Kr the shift keys and Km the mixing keys and the outputs are the A,B,C,D (integers)
- The next function is to return a string of length equal to desired length. when we are going to return the result we need to transform our blocks from integers to string. However magma will cut off some non-meaningful zeros from the left side. So we need to pad each block with zeros for which the hexadecimal representation does not achieve 8 as length.

Results

The block cipher consists of two paired algorithms, one for encryption, E, and the other for decryption,

- CAST256_Encryption(KEY,PT) for encrypting
- CAST256_Decryption(KEY,CT) for decrypting

The decryption algorithm is defined to be the inverse function of encryption. PT is called the plaintext, and CT is termed the cipher text. In our implementation, the encrypted function takes the KEY as a string of bytes in hexadecimal form of length 32, PT as a string in hexadecimal form and the output CT, a decrypted string in hexadecimal form and the decryption function takes the KEY as a string in hexadecimal form, CT as a string in hexadecimal form and the output PT, a decrypted string in hexadecimal form. As mentioned earlier, the CAST 256 algorithm takes the variable key size of 128, 160, 192, 224 or 256 bits. CAST-256 is a symmetric-key algorithm so it uses the same key for both encryption of plaintext and decryption of ciphertext. We included a procedure/function for both encryption and decryption using the paper's test vectors of the following key sizes of 128, 192 and 256. We run the function with the command time testCase(). It executes the function and returns the time it takes to execute this. The screenshot displays the results received. Please note the expected cipher text for comparison with the result of the evaluated cipher text after encryption is done. The two values are the same as can be seen. Further down, the decrypted text is indeed the same as the plaintext input for encryption.



```
> load "/Users/User1/Desktop/CryptoProject/CAST256implementation.txt";
Loading "/Users/User1/Desktop/CryptoProject/CAST256implementation.txt"
> time testCase();


First test (key=128 bits):
Key: 2342bb9efa38542c0af75647f29f615d
Plaintext: 00000000000000000000000000000000
Expected Ciphertext: c842a08972b43d20836c91d1b7530f6b
Evaluated Ciphertext: C842A08972B43D20836C91D1B7530F6B
Successful encryption: true
Decryption phase
Decrypted text: 00000000000000000000000000000000
Successful decryption: true


Second test (key=192 bits):
Key: 2342bb9efa38542cbed0ac83940ac298bac77a7717942863
Plaintext: 00000000000000000000000000000000
Expected Ciphertext: 1b386c0210dcadcbdd0e41aa08a7a7e8
Evaluated Ciphertext: 1B386C0210DCADCBDD0E41AA08A7A7E8
Successful encryption: true
Decryption phase
Decrypted text: 00000000000000000000000000000000
Successful decryption: true


Third test (key=256 bits):
Key: 2342bb9efa38542cbed0ac83940ac2988d7c47ce264908461cc1b5137ae6b604
Plaintext: 00000000000000000000000000000000
Expected Ciphertext: 4f6a2038286897b9c9870136553317fa
Evaluated Ciphertext: 4F6A2038286897B9C9870136553317FA
Successful encryption: true
Decryption phase
Decrypted text: 00000000000000000000000000000000
Successful decryption: true
Time: 0.150
```

References

https://en.wikipedia.org/wiki/CAST-256

https://www.ime.usp.br/~rt/cast256/CAST-256.pdf

http://www.quadibloc.com/crypto/co040410.htm

http://en.citizendium.org/wiki/CAST_(cipher)

https://tools.ietf.org/html/rfc2612