** V I T**

**U N I V E R S I T Y**

(Estd. u/s 3 of UGC Act 1956)

Vellore - 632 014, Tamil Nadu, India

School of Information Technology & Engineering

B.Tech – Information Technology

**ITE320 Network and Information Security**

**CAST 256 Cryptosystem**

**(13BIT0180)**

**Algorithm**

In cryptography, CAST-256 (or CAST6) is a block cipher published in June 1998. It was submitted as a candidate for the Advanced Encryption Standard (AES); however, it was not among the five AES finalists. It is an extension of an earlier cipher, CAST-128; both were designed according to the "CAST" design methodology invented by Carlisle Adams and Stafford Tavares. Howard Heys and Michael Wiener also contributed to the design.CAST-256 uses the same elements as CAST-128, including S-boxes, but is adapted for a block size of 128 bits - twice the size of its 64-bit predecessor. (A similar construction occurred in the evolution of RC5 into RC6). Acceptable key sizes are 128, 160, 192, 224 or 256 bits. CAST-256 is composed of 48 rounds ,sometimes described as 12 "quad-rounds", arranged in a generalized Feistel network. In RFC 2612, the authors state that, "The CAST-256 cipher described in this document is available worldwide on a royalty-free and license-free basis for commercial and non-commercial uses.

The Algorithm Implements Symmetric (Secret) Key Cryptography

The Algorithm is a Block Cipher

**Logic/design elements used in the algorithm**

**Notation**

The following notation is employed in the specification of CAST-256.

Let f1, f2, f3 be as defined for CAST-128.

Let BETA = (ABCD) be a 128-bit block where A, B, C and D are each

32 bits in length.

Let "BETA <- Qi(BETA)" be short-hand notation for the following:

C = C ^ f1(D, Kr0\_(i), Km0\_(i))

B = B ^ f2(C, Kr1\_(i), Km1\_(i))

A = A ^ f3(B, Kr2\_(i), Km2\_(i))

D = D ^ f1(A, Kr3\_(i), Km3\_(i))

Let "BETA <- QBARi(BETA)" be short-hand notation for the

following:

D = D ^ f1(A, Kr3\_(i), Km3\_(i))

A = A ^ f3(B, Kr2\_(i), Km2\_(i))

B = B ^ f2(C, Kr1\_(i), Km1\_(i))

C = C ^ f1(D, Kr0\_(i), Km0\_(i))

(Q(\*) is called a "forward quad-round" and QBAR(\*) is called a

"reverse quad-round".)

Let Kr\_(i) = {Kr0\_(i), Kr1\_(i), Kr2\_(i), Kr3\_(i)} be the set of

rotation keys for the ith quad-round, where Krj\_(i) is a 5-bit

rotation key for f1, f2, or f3 (as specified above).

Let Km\_(i) = {Km0\_(i), Km1\_(i), Km2\_(i), Km3\_(i)} be the set of

masking keys for the ith quad-round, where Kmj\_(i) is a 32-bit

masking key for f1, f2, or f3 (as specified above).

Let KAPPA = (ABCDEFGH) be a 256-bit block where A, B, ..., H are

each 32 bits in length.

Let "KAPPA <- Wi(KAPPA)" be short-hand notation for the

following:

G = G ^ f1(H, Tr0\_(i), Tm0\_(i))

F = F ^ f2(G, Tr1\_(i), Tm1\_(i))

E = E ^ f3(F, Tr2\_(i), Tm2\_(i))

D = D ^ f1(E, Tr3\_(i), Tm3\_(i))

C = C ^ f2(D, Tr4\_(i), Tm4\_(i))

B = B ^ f3(C, Tr5\_(i), Tm5\_(i))

A = A ^ f1(B, Tr6\_(i), Tm6\_(i))

H = H ^ f2(A, Tr7\_(i), Tm7\_(i))

(W(\*) is called a "forward octave".)

Let "Kr\_(i) <- KAPPA" be short-hand notation for the

following: Kr0\_(i) = 5LSB(A), Kr1\_(i) = 5LSB(C), Kr2\_(i) =

5LSB(E), Kr3\_(i) = 5LSB(G)

where 5LSB(x) denotes "the five least significant bits of x".

Let "Km\_(i) <- KAPPA" be short-hand notation for the following:

Km0\_(i) = H, Km1\_(i) = F, Km2\_(i) = D, Km3\_(i) = B

**The CAST-256 Cipher**

BETA = 128bits of plaintext.

for (i=0; i<6; i++)

BETA <- Qi(BETA)

for (i=6; i<12; i++)

BETA <- QBARi(BETA)

128bits of ciphertext = BETA

Round Key Re-Ordering for Decryption

The cipher employs a 256-bit primary key K. Decryption is

identical to encryption except that the sets of quad-round keys

Kr\_(i), Km\_(i) derived from K are used in reverse order as

follows.

for (i=0; i<12; i++)

{

KrNEW\_(i) = Kr\_(11-i)

KmNEW\_(i) = Km\_(11-i)

}

**The CAST-256 Key Schedule**

Initialization:

Cm = 2\*\*30 \* SQRT(2) = 5A827999 (base 16)

Mm = 2\*\*30 \* SQRT(3) = 6ED9EBA1 (base 16)

Cr = 19

Mr = 17

for (i=0; i<24; i++)

{

for (j=0; j<8; j++)

{

Tmj\_(i) = Cm

Cm = (Cm + Mm) mod 2\*\*32

Trj\_(i) = Cr

Cr = (Cr + Mr) mod 32

}

}

**Key Schedule:**

KAPPA = ABCDEFGH = 256 bit of primary key, K.

for (i=0; i<12; i++)

{

KAPPA <- W2i(KAPPA)

KAPPA <- W2i+1(KAPPA)

Kr\_(i) <- KAPPA

Km\_(i) <- KAPPA

}

Note: (|K| = 128) => (E = F = G = H = 0)

(|K| = 160) => (F = G = H = 0)

(|K| = 192) => (G = H = 0)

(|K| = 224) => (H = 0)

**Strength of the algorithm**

-CAST-256 uses the same elements as CAST-128, including [S-boxes](https://en.wikipedia.org/wiki/S-box), but is adapted for a [block size](https://en.wikipedia.org/wiki/Block_size_%28cryptography%29) of 128 bits – twice the size of its 64-bit predecessor.

-Acceptable [key sizes](https://en.wikipedia.org/wiki/Key_size) are 128, 160, 192, 224 or 256 bits. CAST-256 is composed of 48 rounds, sometimes described as 12 "quad-rounds"

-The Algorithm Implements Symmetric (Secret) Key Cryptography

-CAST-256 inherits the strength of the round function.

-performance is quite good (2/3 that of CAST-128)

-code size and complexity are reasonable

-multiple key sizes supported (without any change in performance)

-multiple block sizes may also be specified

**Weakness of the algorithm**

minor weaknesses have been found.

Non-surjective attack, HOD attack but nothing extendable beyond 5-6 rounds

**Attacks possible**

No attacks are possible in this encryption algorithm.

**Attacks impossible**

**Known Plaintext attack** :Cast 256 is provably immune to linear cryptanalysis attack.

**Chosen plaintext only** :Cast 256 is immune to differential analysis with any number of texts.

**Chosen Key Attack**:CAST-256 appears to be secure with respect to this attack. The use of a cipher (built

around the CAST-128 set of round functions) as a key schedule gives confidence that no

exploitable statistical correlation exists between the primary key and the set of generated

round keys. Thus, allowing an attacker to choose a particular primary key difference

appears to yield no exploitable similarities in the corresponding sets of round keys

compared with the victim encrypting with two randomly-chosen primary keys

**Related Key Attack**:CAST-256 appears to be secure with respect to this attack. The use of a cipher (built

around the CAST-128 set of round functions) as a key schedule gives confidence that no

exploitable statistical correlations exist within the set of generated round keys. Thus, this

attack, which depends upon the use of a simple derivation algorithm for a round key from

previous round keys, appears not to be applicable to CAST-256.

**Combination Attack**: This cipher is immune to the combination attacks currently known in the literature.

**CAST-256 (which has 48 rounds and uses the CAST-128 round functions) is immune to a higher-order differential attack**.

**Benefits**

The CAST-256 cipher has a number of advantages compared with other algorithms found

in the open literature, including the following.

• Speed: the cipher has very good encryption / decryption performance and an

acceptable key set-up time for most environments.

• History: the s-boxes and round functions have been examined in detail by a number

of cryptographers and cryptanalysts in the context of CAST-128.

• Simplicity: the operations used in the cipher (XOR, addition, subtraction, rotation)

are all simple, available, and fast on typical computing platforms.

• Identical Operation: encryption and decryption are identical operations, requiring a

simple reversal in the order of the round keys.

• Fixed Speed: the encryption / decryption speed is unaffected by a change in key size

(from 128 bits to 256 bits).

• Secure: quite conservative analysis indicates that the cipher is as strong as its key

size.

**Drawbacks**

48 rounds is a lot of rounds

**Computational power and other resources required**

For environments in which memory is not a scarce resource, s-boxes

S1 and S2 can be combined into three 16 32 s-boxes (one corresponding to S1 S2, one corresponding to S1 - S2 , and one corresponding to S1 + S2 , for each of the three round function types). This saves one memory lookup and combining operation per round, which will result in a modest performance increase.

**Limitations of the system, Research issues and research gaps identified, features\properties\functionalities missing**

Limitations

For some specific environments, the following may be seen as limitations of the CAST-

256 cipher.

• Memory: the 4 Kilobytes of total storage required for the CAST-256 s-boxes may be

too high for some environments with very constrained resources.

• Key Set-Up Time: the time to generate the set of round keys from the primary key

(four times the time required to encrypt a single block of data), although comparable

to DES, may be too slow for some very high-speed environments that need to change

keys very frequently.

• Rotation Operation: the key-dependent rotation operation in the CAST-256 round

function may be too slow for some environments that cannot do a multi-bit rotation in

a single machine instruction.

**Application**

CAST-128, also known as CAST5, is a block cipher used in a number of products, notably as the default cipher in some versions of GNU Privacy Guard (GPG) and Pretty Good Privacy (PGP) systems. It has also been approved for Canadian government use by the Communications Security Establishment.

CAST-256, also known as CAST6, was derived from CAST-128 and published in June 1998. It was submitted as a candidate for the Advanced Encryption Standard (AES), however it was not among the five AES finalists.CAST-256 uses the same elements as CAST-128

The CAST-256 cipher is available worldwide on a royalty-free and licence-free basis for commercial and non-commercial uses.

**Suggestions/proposal for Extension/Enhancement/Improvement/Modification of this Cryptosystem**

Decrease the number of rounds because it makes the algorithm slow

**Program: C/C++/Java code**

import java.security.Provider;

import javax.crypto.Cipher;

import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.codec.binary.Hex;

import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class Cast6 {

static final String KEY\_ALGO = "CAST6";

static final String CIPHER\_ALGO = "CAST6/ECB/NOPADDING";

static String keytext = "2342bb9efa38542c0af75647f29f615d";

static String plaintext = "00000000000000000000000000000000";

static String ciphertext = "c842a08972b43d20836c91d1b7530f6b";

static Provider bc = new BouncyCastleProvider();

public static void main(String[] args) throws Exception {

System.out.println("encrypting");

String actual = encrypt();

System.out.println("actual: " + actual);

System.out.println("expect: " + ciphertext);

System.out.println("decrypting");

actual = decrypt();

System.out.println("actual: " + actual);

System.out.println("expect: " + plaintext);

}

static String encrypt() throws Exception {

Cipher cipher = Cipher.getInstance(CIPHER\_ALGO, bc);

byte[] keyBytes = Hex.decodeHex(keytext.toCharArray());

SecretKeySpec key = new SecretKeySpec(keyBytes, KEY\_ALGO);

cipher.init(Cipher.ENCRYPT\_MODE, key);

byte[] input = Hex.decodeHex(plaintext.toCharArray());

byte[] output = cipher.doFinal(input);

String actual = Hex.encodeHexString(output);

return actual;

}

static String decrypt() throws Exception {

Cipher cipher = Cipher.getInstance(CIPHER\_ALGO, bc);

byte[] keyBytes = Hex.decodeHex(keytext.toCharArray());

SecretKeySpec key = new SecretKeySpec(keyBytes, KEY\_ALGO);

cipher.init(Cipher.DECRYPT\_MODE, key);

byte[] output = cipher.doFinal(Hex.decodeHex(ciphertext.toCharArray()));

String actual = Hex.encodeHexString(output);

return actual;

}

}