The CAST-128 Encryption Algorithm

Status of this Memo

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

There is a need in the Internet community for an unencumbered encryption algorithm with a range of key sizes that can provide security for a variety of cryptographic applications and protocols.

This document describes an existing algorithm that can be used to satisfy this requirement. Included are a description of the cipher and the key scheduling algorithm (Section 2), the s-boxes (Appendix A), and a set of test vectors (Appendix B).

TABLE OF CONTENTS

STATUS OF THIS MEMO................................................1
ABSTRACT..............................................................1
1. INTRODUCTION....................................................1
2. DESCRIPTION OF ALGORITHM.....................................2
3. INTELLECTUAL PROPERTY CONSIDERATIONS....................8
4. SECURITY CONSIDERATIONS.......................................8
5. REFERENCES......................................................8
6. AUTHOR’S ADDRESS..............................................8
APPENDICES
A. S-BOXES.........................................................9
B. TEST VECTORS................................................15

1. Introduction

This document describes the CAST-128 encryption algorithm, a DES-like Substitution-Permutation Network (SPN) cryptosystem which appears to have good resistance to differential cryptanalysis, linear cryptanalysis, and related-key cryptanalysis. This cipher also possesses a number of other desirable cryptographic properties, including avalanche, Strict Avalanche Criterion (SAC), Bit Independence Criterion (BIC), no complementation property, and an absence of weak and semi-weak keys. It thus appears to be a good
candidate for general-purpose use throughout the Internet community wherever a cryptographically-strong, freely-available encryption algorithm is required.

Adams [Adams] discusses the CAST design procedure in some detail; analyses can also be obtained on-line (see, for example, [Web1] or [Web2]).

2. Description of Algorithm

CAST-128 belongs to the class of encryption algorithms known as Feistel ciphers; overall operation is thus similar to the Data Encryption Standard (DES). The full encryption algorithm is given in the following four steps.

INPUT: plaintext m1...m64; key K = k1...k128.
OUTPUT: ciphertext c1...c64.

1. (key schedule) Compute 16 pairs of subkeys \{Kmi, Kri\} from K (see Sections 2.1 and 2.4).
2. \((L0,R0) \leftarrow (m1...m64)\). (Split the plaintext into left and right 32-bit halves \(L0 = m1...m32\) and \(R0 = m33...m64\).)
3. (16 rounds) for i from 1 to 16, compute \(L_i\) and \(R_i\) as follows:
   \(L_i = R_{i-1}\);
   \(R_i = L_{i-1} \oplus f(R_{i-1}, Kmi, Kri)\), where \(f\) is defined in Section 2.2
   \((f\) is of Type 1, Type 2, or Type 3, depending on i).
4. \(c1...c64 \leftarrow (R_{16}, L_{16})\). (Exchange final blocks \(L_{16}, R_{16}\) and concatenate to form the ciphertext.)

Decryption is identical to the encryption algorithm given above, except that the rounds (and therefore the subkey pairs) are used in reverse order to compute \((L0,R0)\) from \((R_{16}, L_{16})\).

See Appendix B for test vectors which can be used to verify correctness of an implementation of this algorithm.

2.1. Pairs of Round Keys

CAST-128 uses a pair of subkeys per round: a 32-bit quantity \(Km\) is used as a "masking" key and a 5-bit quantity \(Kr\) is used as a "rotation" key.
2.2. Non-Identical Rounds

Three different round functions are used in CAST-128. The rounds are as follows (where "D" is the data input to the f function and "Ia" - "Id" are the most significant byte through least significant byte of I, respectively). Note that "+" and "-" are addition and subtraction modulo $2^{32}$, "^" is bitwise XOR, and "<<<" is the circular left-shift operation.

- **Type 1:**  $I = ((Kmi + D) <<< Kri)$
  $f = ((S1[Ia] ^ S2[Ib]) - S3[Ic]) + S4[Id]$

- **Type 2:**  $I = ((Kmi ^ D) <<< Kri)$
  $f = ((S1[Ia] - S2[Ib]) + S3[Ic]) ^ S4[Id]$

- **Type 3:**  $I = ((Kmi - D) <<< Kri)$
  $f = ((S1[Ia] + S2[Ib]) ^ S3[Ic]) - S4[Id]$

Rounds 1, 4, 7, 10, 13, and 16 use f function Type 1. Rounds 2, 5, 8, 11, and 14 use f function Type 2. Rounds 3, 6, 9, 12, and 15 use f function Type 3.

2.3. Substitution Boxes

CAST-128 uses eight substitution boxes: s-boxes S1, S2, S3, and S4 are round function s-boxes; S5, S6, S7, and S8 are key schedule s-boxes. Although 8 s-boxes require a total of 8 KBytes of storage, note that only 4 KBytes are required during actual encryption / decryption since subkey generation is typically done prior to any data input.

See Appendix A for the contents of s-boxes S1 - S8.

2.4. Key Schedule

Let the 128-bit key be x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF, where x0 represents the most significant byte and xF represents the least significant byte.

Let z0..zF be intermediate (temporary) bytes.

Let Si[] represent s-box i and let "^" represent XOR addition.
The subkeys are formed from the key $x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF$ as follows.

$$
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8] 
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA] 
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9] 
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB] 
$$

$$
K1 = S5[z8] ^ S6[z9] ^ S7[z7] ^ S8[z6] ^ S5[z2] 
K3 = S5[zC] ^ S6[zD] ^ S7[z3] ^ S8[z2] ^ S7[z9] 
K4 = S5[zE] ^ S6[zF] ^ S7[z1] ^ S8[z0] ^ S8[zC] 
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0] 
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2] 
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1] 
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3] 
K5 = S5[x3] ^ S6[x2] ^ S7[xC] ^ S8[xD] ^ S5[x8] 
K6 = S5[x1] ^ S6[x0] ^ S7[xE] ^ S8[xF] ^ S6[xD] 
K7 = S5[x7] ^ S6[x6] ^ S7[x8] ^ S8[x9] ^ S7[x3] 
K8 = S5[x5] ^ S6[x4] ^ S7[xA] ^ S8[xB] ^ S8[x7] 
$$

$$
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8] 
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA] 
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9] 
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB] 
$$

$$
K9 = S5[z3] ^ S6[z2] ^ S7[zC] ^ S8[zD] ^ S5[z9] 
K10 = S5[z1] ^ S6[z0] ^ S7[zE] ^ S8[zF] ^ S6[zC] 
K11 = S5[z7] ^ S6[z6] ^ S7[z8] ^ S8[z9] ^ S7[z2] 
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0] 
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2] 
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1] 
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3] 
K13 = S5[x8] ^ S6[x9] ^ S7[x7] ^ S8[x6] ^ S5[x3] 
K14 = S5[xA] ^ S6[xB] ^ S7[x5] ^ S8[x4] ^ S6[x7] 
K15 = S5[xC] ^ S6[xD] ^ S7[x3] ^ S8[x2] ^ S7[x8] 
K16 = S5[xE] ^ S6[xF] ^ S7[x1] ^ S8[x0] ^ S8[xD]
2.4.1. Masking Subkeys And Rotate Subkeys

Let Km1, ..., Km16 be 32-bit masking subkeys (one per round).
Let Kr1, ..., Kr16 be 32-bit rotate subkeys (one per round); only the least significant 5 bits are used in each round.

for (i=1; i<=16; i++) { Km[i] = Ki; Kri[i] = K16+i; }
2.5. Variable Keysize

The CAST-128 encryption algorithm has been designed to allow a key size that can vary from 40 bits to 128 bits, in 8-bit increments (that is, the allowable key sizes are 40, 48, 56, 64, ..., 112, 120, and 128 bits). For variable keysize operation, the specification is as follows:

1) For key sizes up to and including 80 bits (i.e., 40, 48, 56, 64, 72, and 80 bits), the algorithm is exactly as specified but uses 12 rounds instead of 16;

2) For key sizes greater than 80 bits, the algorithm uses the full 16 rounds;

3) For key sizes less than 128 bits, the key is padded with zero bytes (in the rightmost, or least significant, positions) out to 128 bits (since the CAST-128 key schedule assumes an input key of 128 bits).

Note that although CAST-128 can support all 12 key sizes listed above, 40 bits, 64 bits, 80 bits, and 128 bits are the sizes that find utility in typical environments. Therefore, it will likely be sufficient for most implementations to support some subset of only these four sizes.

In order to avoid confusion when variable keysize operation is used, the name CAST-128 is to be considered synonymous with the name CAST5; this allows a keysize to be appended without ambiguity. Thus, for example, CAST-128 with a 40-bit key is to be referred to as CAST5-40; where a 128-bit key is explicitly intended, the name CAST5-128 should be used.

2.6. CAST5 Object Identifiers

For those who may be using CAST in algorithm negotiation within a protocol, or in any other context which may require the use of OBJECT IDENTIFIERS, the following OIDs have been defined.

```
algorithms OBJECT IDENTIFIER ::= { iso(1) memberBody(2) usa(840) nt(113533) nsn(7) algorithms(66) }
```
cast5CBC OBJECT IDENTIFIER ::= { algorithms cast5CBC(10) }

Parameters ::= SEQUENCE {
    iv OCTET STRING DEFAULT 0, -- Initialization vector
    keyLength INTEGER            -- Key length, in bits
}

Note: The iv is optional and defaults to all-zero. On the encoding end, if an all-zero iv is used, then it should be absent from the Parameters. On the decoding end, an absent iv should be interpreted as meaning all-zeros.

This is encryption and decryption in CBC mode using the CAST-128 symmetric block cipher algorithm.

cast5MAC OBJECT IDENTIFIER ::= { algorithms cast5MAC(11) }

Parameters ::= SEQUENCE {
    macLength INTEGER,      -- MAC length, in bits
    keyLength INTEGER        -- Key length, in bits
}

This is message authentication using the CAST-128 symmetric block cipher algorithm.

pbeWithMD5AndCast5CBC OBJECT IDENTIFIER ::= {
    algorithms pbeWithMD5AndCAST5-CBC(12) }

Parameters ::= SEQUENCE {
    salt OCTET STRING,
    iterationCount INTEGER, -- Total number of hash iterations
    keyLength INTEGER       -- Key length, in bits
}

Note: The IV is derived from the hashing procedure and therefore need not be included in Parameters.

This is password-based encryption and decryption in CBC mode using MD5 and the CAST-128 symmetric block cipher. See PKCS #5 (which uses the DES cipher) for details of the PBE computation.
2.7. Discussion

CAST-128 is a 12- or 16-round Feistel cipher that has a blocksize of 64 bits and a keysize of up to 128 bits; it uses rotation to provide intrinsic immunity to linear and differential attacks; it uses a mixture of XOR, addition and subtraction (modulo $2^{32}$) in the round function; and it uses three variations of the round function itself throughout the cipher. Finally, the 8x32 s-boxes used in the round function each have a minimum nonlinearity of 74 and a maximum entry of 2 in the difference distribution table.

This cipher appears to have cryptographic strength in accordance with its keysize (128 bits) and has very good encryption / decryption performance: 3.3 MBytes/sec on a 150 MHz Pentium processor.

3. Intellectual Property Considerations

The CAST-128 cipher described in this document is available worldwide on a royalty-free basis for commercial and non-commercial uses.

4. Security Considerations

This entire memo is about security since it describes an algorithm which is specifically intended for cryptographic purposes.

5. References


6. Author’s Address

Carlisle Adams
Entrust Technologies
750 Heron Road,
Ottawa, Canada, K1V 1A7

E-mail: cadams@entrust.com
Phone: +1.613.763.9008
## Appendix A. S-Boxes

### S-Box S1

<table>
<thead>
<tr>
<th>S1</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>30fb40d4</td>
<td>9fa0ff0b 6becc2d2f 3f258c7a 1e213f2f 9c004dd3 603e540 cf9fc949</td>
</tr>
<tr>
<td>bfd4a4f7</td>
<td>88bbbd5b e2034090 98d09675 663a0e0 15c361d2 c2e7661d 22d4ff8e</td>
</tr>
<tr>
<td>28683b6f</td>
<td>c07fd059 ff2397c8 75f50e2 43c340d3 df2f8656 887ca41a 2a2dbd2d</td>
</tr>
<tr>
<td>a1c9e0d6</td>
<td>346c4819 61b768d7 22540f2f 2abe32e1 aa54166b 22b0c054 bc8e5935</td>
</tr>
<tr>
<td>66db40c8</td>
<td>979439f2 004dff2f 2db9d2de 93b159f b4d8ee11 4bff345d f01144f9</td>
</tr>
<tr>
<td>b82cbeaf</td>
<td>d751d159 6ff70f0ed 5581685 d5b1caad a1ac2d2e 2a4b769c 19b0c50</td>
</tr>
<tr>
<td>4b6d2f7f</td>
<td>30fb40d4 6becc2d2f 3f258c7a 1e213f2f 9c004dd3 603e540 cf9fc949</td>
</tr>
<tr>
<td>fd45c240</td>
<td>30fb40d4 6becc2d2f 3f258c7a 1e213f2f 9c004dd3 603e540 cf9fc949</td>
</tr>
</tbody>
</table>

### S-Box S2

<table>
<thead>
<tr>
<th>S2</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1f201094</td>
<td>ef0ba75b 69e3c7f9 3f934380 6f61cf7a eec5207a 55889c94 72fc0651</td>
</tr>
<tr>
<td>ada7ef79</td>
<td>41ed7235 d5a63ce de0436ba 99c430ef 5f0c0794 1bdcc87bd a1d6eef3</td>
</tr>
<tr>
<td>a0b5f27f</td>
<td>59e83605 ee15b094 9fffd909 dc440086 ef944459 ba83b3c3 e0c3cfdb</td>
</tr>
<tr>
<td>d1da4181</td>
<td>3b092a1b f997f1c5 a5e6c7f8b 01420d0d eae7f5b7 25a1ff4f1 e180f806</td>
</tr>
<tr>
<td>1fc4080</td>
<td>179ae0b7 d37ac6a9 fe5830a4 98de8b7f 77e83f4e 7992296 24fa9f7b</td>
</tr>
<tr>
<td>e113c85b</td>
<td>acc40083 d7503525 f7eaa15f 62143154 0d55a6b3 5d681121 c866c359</td>
</tr>
<tr>
<td>3d63cf73</td>
<td>cee234c6 d48de78 5c72b2a1 071f6181 39f7627f 361e0384 e4e653b7</td>
</tr>
<tr>
<td>602f64a4</td>
<td>d5ac9dc1 1bca4653 9e81032d 2701f50c 99847ab4 a0e3df79 ba6c538c</td>
</tr>
<tr>
<td>10843094</td>
<td>2537a9e5 f646fffe a1ff3b1f 208c6f6a 8f458c74 d9e0a227 4ec73a34</td>
</tr>
<tr>
<td>fc8846f9</td>
<td>3e4de8df ef050e88 3559648d 8a45388c 1d804366 721d9b7f d5a6c252 e49754bd</td>
</tr>
</tbody>
</table>
S-Box S4

```
a2048016 97573833 d7207d67 de0f8f3d 72f87b33 abcc4f33 7688c55d 7b00a6b0
947b0001 570075d2 f9b88f8 8942019e 4264a5ff 85630e20 72dbd92b ee971b69
6ea22fde 5f08ae2b af7a616d 5c98767 cf1febd2 61efc8c2 f1ac2571 cc8239c2
67214cb8 b1e583d1 b7dc3e62 7f10bdce f90a5c38 0ff0443d 606e6dc6 60543a49
527c148 2be98a1d 8ab41738 20e1be24 af96da0f 0fca5fde 71ff904c 2d195ffe
28f9350 8334b362 d9112b0 2b6da80 64b1e31 9c305a00 52cbce88 1b0358a8
f7baefd5 41a2ed9c a4315c11 83323ec5 dfef4636 a133c501 e9d3531c ee353783

S-Box S5

```
9db30420 1fb6e9de a7be7bef d27701f b28e5e5f 0df80a5 8644213e 2e097fd2
8644213e 2e097fd2 0df80a5 8644213e 2e097fd2 0df80a5 8644213e 2e097fd2
```
S-Box S7

S-Box S7

Adams                        Informational                     [Page 13]
S-Box S8

e216300d bbddfffc a7ebdabd 35648095 7789f8b7 e6c1121b 0e241600 052ce8b5
11a3cfb0 e5952f11 ece7990a 9386d174 2a42931c 7e381111 b12def3a 37dddfc
de9adeb1 0a0cc32c be197029 84a00940 bb243a0f b4d137cf b4e79f00 049eedfd
0b15a15d 480d3168 8b8be645 a7ebdabd 35648095 7789f8b7 e6c1121b 0e241600 052ce8b5
11a9cfb0 e5952f11 ece7990a 9386d174 2a42931c 7e381111 b12def3a 37dddfc
94074251 5c7dca40 aabbe66d aa40216c 60b15a65 0d29243a 1b7029e5 0000ff00 11a9cfb0 e5952f11 ece7990a 9386d174 2a42931c 7e381111 b12def3a 37dddfc
12a8eddc fda3335d 176f43e8 71f46b4d 38120922 ce99ad4e b48769ad 965bd862
82f3d055 66fb9767 15b80b4e 1d5b47a0 4cfde06f c82ec4b8 57e8726e 647a78fc
99865d44 608bd593 6c200e03 39dc5ff6 5d0b00a3 ae63aff2 7e86bd32 70108c0c
b8d35049 99c64a2a 96b6f491 9e7ed5d3 06918548 58b7e07e 0274e2f2 e522ffeb 1d24708c 1c7e27cd 04eb215b 3cf1d2e2 19b47a38 424f7618 35856039
9d17ade7 27eb35e6 c9aff67b 36baf5b8 09c467cd c18910b1 e11d0f7b 06d1af8
7107c608 2d6353e4 dd4e495c 64c6d006 b40c62c 3dd00db3 708f8f34 77d51d42
264f620f 2b4b2f7d 15c1b79e 46a52564 f8d7e54e 3e378160 7895ca5 859c15a5
e6459788 c37bc75f db0b7a0c 0676a3ab 7f2b9b1e 31842e7b 24529rf7 8bfe472
835f4c8 6df4cf02 96fb5195 fda0af0c b0fe134c e2506d3d 4f9b12ea f215f225
a223736f 9bf4c428 25d04979 34c713f8 c418187e ea76e987 071d6f42 522ffeb1 d24708c 1c7e27cd 04eb215b 3cf1d2e2 19b47a38 424f7618 35856039
5e9af270 a04a441 3c7f8c99 92eebc8e dd67016d 51682eb
a842eedf fdb60a4 1907b75 20e30280 24d8c29 02b3822e a842bfa 6fbc428 5b1d42
b6f2cf3b 9f326442 ch15a4cc 0b1a4504 f1e47d8d 844a1be5 ba8f7d7c 42c8a70
3d7e4a0e 6e85b7a7 d53f5af6 20cfc48c cca4d428 79d130a4 34b6efb 33d3cddc
7785b357 37eefc5b c5068788 e580b36e 4e68b8f4 c508b37e e08099ea 39e8ebc7
1324af94 43b7950c 2fee7d1c 23b613bd dd06caca 32f932b 0b424829 acf3ebc3
5715f6b7 eb3f78dd f267616f c140cee4 9052815e 5e410f9b a48a2485 2eda7fa4
e87b40e4 e98ea0f8 588989ef efd390fc dd0735db db856954 38d7e5b2 57720101
730edefb c5643113 9491e4f 503c23fa 6b461722 5723d2a4 e0776959 f9c1a87f
7a5b2121 d187b80e 292634d ba510c9f 81f7e479f ad1163ed ea7b5965 7aa0726e
11403092 00da6d77 4a0cd61f ad1f4603 605bdf00 9eed3c64 2b3e7d28 a0e736a0 556a6b9b 10853209 c7eb8f37 2de705ca 89515070 df09822b bd6916ac
aa124ef2 87451c0f e0f6a27a 3ada4819 4cf1764f 0d771c2b 67c8b156 350d8384
5938a0f 42399ef3 639b70b7 0e84093d 4a9a3e61 8360d878 1a9ab0c 1149382c
e97625a5 06141db7 025b244b 0c768347 589e8d82 02d59d1 a466bb1e 84b9a82c
04f19130 ba6e4ec0 99265164 1ee7230d 50b2ad80 eae6801 8db2a283 ea8bf59e

Adams                        Informational                     [Page 14]
Appendix B. Test Vectors

This appendix provides test vectors for the CAST-128 cipher described in this document.

B.1. Single Plaintext-Key-Ciphertext Sets

In order to ensure that the algorithm is implemented correctly, the following test vectors can be used for verification (values given in hexadecimal notation).

128-bit key         = 01 23 45 67 12 34 56 78 23 45 67 89 34 56 78 9A
plaintext   = 01 23 45 67 89 AB CD EF
ciphertext  = 23 8B 4F E5 84 7E 44 B2

80-bit  key         = 01 23 45 67 12 34 56 78 23 45 00 00 00 00 00 00
plaintext   = 01 23 45 67 89 AB CD EF
ciphertext  = EB 6A 71 1A 2C 02 27 1B

40-bit  key         = 01 23 45 67 12
plaintext   = 01 23 45 67 89 AB CD EF
ciphertext  = 7A C8 16 D1 6E 9B 30 2E

B.2. Full Maintenance Test

A maintenance test for CAST-128 has been defined to verify the correctness of implementations. It is defined in pseudo-code as follows, where a and b are 128-bit vectors, aL and aR are the leftmost and rightmost halves of a, bL and bR are the leftmost and rightmost halves of b, and encrypt(d,k) is the encryption in ECB mode of block d under key k.

Initial a = 01 23 45 67 12 34 56 78 23 45 67 89 34 56 78 9A (hex)
Initial b = 01 23 45 67 12 34 56 78 23 45 67 89 34 56 78 9A (hex)

do 1,000,000 times
{
    aL = encrypt(aL,b)
    aR = encrypt(aR,b)
    bL = encrypt(bL,a)
    bR = encrypt(bR,a)
}

Verify a == EE A9 D0 A2 49 FD 3B A6 B3 43 6F B8 9D 6D CA 92 (hex)
Verify b == B2 C9 5E B0 0C 31 AD 71 80 AC 05 B8 E8 3D 69 6E (hex)