The CAST-256 Encryption Algorithm
Carlisle Adams

CAST-256 is a symmetric cipher designed in accordance with the CAST design procedure as outlined in [A97]. It is an extension of the CAST-128 cipher and has been submitted as a candidate for NIST’s Advanced Encryption Standard (AES) effort -- see http://csrc.nist.gov/encryption/aes/aes_home.htm for details.

This document contains several sections of the CAST-256 AES Submission Package delivered to NIST on June 9th, 1998. All complete submissions received by NIST will be made public in late August at the First AES Candidate Conference, but the following material is being made available now so that public analysis of the CAST-256 algorithm may begin (see, for example, http://www.ii.uib.no/~larst/aes.html for the current status of submitted algorithms).

Many thanks are due to those who worked with me in the (long, challenging, frustrating, and very enjoyable!) design and analysis phases that ultimately led to the detailed specification given below: Howard Heys (Memorial University); Stafford Tavares (Queen’s University); and Michael Wiener (Entrust Technologies). As well, many thanks are due to the two who did the various implementations on a variety of platforms (Reference C, Optimized C, Optimized Java, and even M6811 Assembler): Serge Mister and Ian Clysdale (both of Entrust Technologies).

CAST-256
Algorithm Specification

1. Algorithm Specification

1.1 CAST-128 Notation

The following notation from CAST-128 [A97b, A97c] is relevant to CAST-256.

- CAST-128 uses a pair of subkeys per round: a 5-bit quantity $k_r^i$ is used as a “rotation” key for round $i$ and a 32-bit quantity $k_m^i$ is used as a “masking” key for round $i$.

- Three different round functions are used in CAST-128. The rounds are as follows (where $D$ is the data input to the operation, $I_0$ - $I_7$ are the most significant byte through least significant byte of $I$, respectively, $S_i$ is the $i^{th}$ s-box (see following page for s-box definitions), and $O$ is the output of the operation). Note that $+$ and $-$ are addition and subtraction modulo $2^{32}$, $\oplus$ is bitwise eXclusive-OR, and $\ll$ is the circular left-shift operation.

Type 1:

$I = ((k_m^i + D) \ll k_r^i)$

$O = ((S_1[I_0] \oplus S_2[I_1]) - S_3[I_2]) + S_4[I_3]$ 

Type 2:

$I = ((k_m^i \oplus D) \ll k_r^i)$

$O = ((S_1[I_0] - S_2[I_1]) + S_3[I_2]) \oplus S_4[I_3]$ 

Type 3:

$I = ((k_m^i - D) \ll k_r^i)$

$O = ((S_1[I_0] + S_2[I_1]) \oplus S_3[I_2]) - S_4[I_3]$ 

Let $f_1, f_2, f_3$ be keyed round function operations of Types 1, 2, and 3 (respectively) above.
CAST-128 Notation (cont'd)

- CAST-128 uses four round function substitution boxes (s-boxes), \( S_1 \) - \( S_4 \). These are defined as follows (entries in hexadecimal notation are to be read left-to-right, top-to-bottom).

**S-Box \( S_1 \)**

<table>
<thead>
<tr>
<th>( S_1 )</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x30fb4d4</td>
<td>9fa0ff0b 6beccdd2f 3f258c7a 1e213f2f 9c004dd3 603e54e0 cf9fc949</td>
</tr>
<tr>
<td>0xb4daf27</td>
<td>86bbbd5b e20340c9 98d09675 6e63a0e0 15c361d2 c2e7661d 22d4ff8e</td>
</tr>
<tr>
<td>0x2863bf6</td>
<td>c07df059 ff2379c8 775f50e2 43c340d3 df2f8656 887ca41a a2d286af</td>
</tr>
<tr>
<td>0x1a9e0d6</td>
<td>346c4819 61b76d87 22540f2f 2abe32e1 aa54166b 22d341d0</td>
</tr>
<tr>
<td>0x6d6b0c8</td>
<td>874392f2 04dfdf2f 2b9d26de 97943fac 497c1d8 52764a87 b5f437a7</td>
</tr>
<tr>
<td>0xb82c0af</td>
<td>d751d159 6ff7f0ed 5a0971af 827b68d0 90ecf52e 22b0c054 bc2e59e6</td>
</tr>
<tr>
<td>0x46b2d7f</td>
<td>50bb64a2 d26469d1 b73239f0 e93b159f 48e8e411 bff3345d</td>
</tr>
<tr>
<td>0xf4d5c2a4</td>
<td>ad3197f3 c4fd602e 55fc8165 d2664910 bee5812d b7332290</td>
</tr>
<tr>
<td>0x8822d4f2</td>
<td>0c6e4f38 a4e4bf7d 4f5ba272 564c1d2f c59c5319 b949e354 b04669fe</td>
</tr>
<tr>
<td>0xb166aba8</td>
<td>c71358dd 6385c545 110f935d 57538ad5 6a390493 e63d37e0 2a54f6b3</td>
</tr>
<tr>
<td>0x3a787df5</td>
<td>6276a0b5 19a6fcd8 7a4220a6 29fd94d5 f61b1891 bb72275e aa508167</td>
</tr>
<tr>
<td>0x389019d1</td>
<td>c6b050e8 84c78c8b 2ad75a0f 874a1427 a2d1936b 2ad286af aa5296d1</td>
</tr>
<tr>
<td>0x7d89436c</td>
<td>452575d9 93b38e2e 187184c9 6c00b32d 73e2bb14 a0bebc3c 54623779</td>
</tr>
<tr>
<td>0x64459eab</td>
<td>3f328b82 7718c8f2 59a2cea6 04ee002e 89fe786e 3fa90950 325ff6c2</td>
</tr>
<tr>
<td>0xb813f065</td>
<td>6e7c35c9 6b63a0f7 19a6fcdf 7a42206a 29fd94d5 f61b1891 bb72275e aa508167</td>
</tr>
</tbody>
</table>

**S-Box \( S_2 \)**

<table>
<thead>
<tr>
<th>( S_2 )</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1f20109d</td>
<td>ef0ba75b 6beccdd2f 3f258c7a 1e213f2f 9c004dd3 603e54e0 cf9fc949</td>
</tr>
<tr>
<td>0xada77f0</td>
<td>61d723e5 d5563ce4 de0436ba 99c4340f 5f0c0794 18dcbdb7 a1d6eff3</td>
</tr>
<tr>
<td>0x0b52f7b</td>
<td>86b38e05 ee1b5094 e9ffdf09 dc440086 ef944459 ba83ccbb e0c3c9fb</td>
</tr>
<tr>
<td>0xda01bf1</td>
<td>30b92a1b f997fc1 a5e6cf7b 01422dd0 e4e7f5bd 25a1ff41 e80f80f6</td>
</tr>
<tr>
<td>0xf140870</td>
<td>179bee7a3 d3cc6a97 fe5830a4 98db8e7f 77e83f4e 79929269 24af97fb</td>
</tr>
<tr>
<td>0xe113c85b</td>
<td>ac042f1c 01137ea 8dbfaadb 35b9a4f0 3526ff2a 3cb4d09c bc306ed9</td>
</tr>
<tr>
<td>0x9852666</td>
<td>5648ff2f ff5e662d 0ce3d63d 7c63c2bf 700b4e51 d5ea50f1 85a92872</td>
</tr>
<tr>
<td>0xaaffbda7</td>
<td>d4234870 a7870bf3 2d3b4d79 42ef0198 c0de0ede7 26470d8b f8818814</td>
</tr>
<tr>
<td>0x47ad077</td>
<td>7c0ce5c5 dc139211 38972b98 f2d5f4db ab388653 6e2f1ee3 b83719ce</td>
</tr>
<tr>
<td>0xbd91e046</td>
<td>9a56545e dc39200c 206c5791 96b2da1c e1696ff8 b141ab08 7c8a98b9</td>
</tr>
<tr>
<td>0x1a69e787</td>
<td>02094832 a2f7c579 429e4f7d 247b169c 5ac90f49 dd8f0f00 5c8165bf</td>
</tr>
</tbody>
</table>

**S-Box \( S_3 \)**

<table>
<thead>
<tr>
<th>( S_3 )</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1f20109d</td>
<td>ef0ba75b 6beccdd2f 3f258c7a 1e213f2f 9c004dd3 603e54e0 cf9fc949</td>
</tr>
<tr>
<td>0xada77f0</td>
<td>61d723e5 d5563ce4 de0436ba 99c4340f 5f0c0794 18dcbdb7 a1d6eff3</td>
</tr>
<tr>
<td>0x0b52f7b</td>
<td>86b38e05 ee1b5094 e9ffdf09 dc440086 ef944459 ba83ccbb e0c3c9fb</td>
</tr>
<tr>
<td>0xda01bf1</td>
<td>30b92a1b f997fc1 a5e6cf7b 01422dd0 e4e7f5bd 25a1ff41 e80f80f6</td>
</tr>
<tr>
<td>0xf140870</td>
<td>179bee7a3 d3cc6a97 fe5830a4 98db8e7f 77e83f4e 79929269 24af97fb</td>
</tr>
<tr>
<td>0xe113c85b</td>
<td>ac042f1c 01137ea 8dbfaadb 35b9a4f0 3526ff2a 3cb4d09c bc306ed9</td>
</tr>
<tr>
<td>0x9852666</td>
<td>5648ff2f ff5e662d 0ce3d63d 7c63c2bf 700b4e51 d5ea50f1 85a92872</td>
</tr>
<tr>
<td>0xaaffbda7</td>
<td>d4234870 a7870bf3 2d3b4d79 42ef0198 c0de0ede7 26470d8b f8818814</td>
</tr>
<tr>
<td>0x47ad077</td>
<td>7c0ce5c5 dc139211 38972b98 f2d5f4db ab388653 6e2f1ee3 b83719ce</td>
</tr>
<tr>
<td>0xbd91e046</td>
<td>9a56545e dc39200c 206c5791 96b2da1c e1696ff8 b141ab08 7c8a98b9</td>
</tr>
<tr>
<td>0x1a69e787</td>
<td>02094832 a2f7c579 429e4f7d 247b169c 5ac90f49 dd8f0f00 5c8165bf</td>
</tr>
</tbody>
</table>
S-Box $S_3$

8defc240 25fa5d9f eb903dbf e810c907 369fe44b 8c1fc644 aececa90
beb1f9bf 8e0b490f 68cb01ee 21f78dfe 7e6db2d2 5f8b9a04 7ed3bcb9
a747d2d0 1651192e af70bf3e 58c31380 0a0fb402 0f7f8ef2 8c96fdad
8c96fdad 5d2c2aae 8ee99a49
2f7e850 d7c07f7e 2d7c3e3c 0f7f8ef2 8c96fdad
9db30420 1fb6e9de a7be7bef d273a298
9069d35e 05c935b0 0f7f8ef2 8c96fdad
47c7ebc2 2d7c3e3c 0f7f8ef2 8c96fdad
8c96fdad 5d2c2aae 8ee99a49

S-Box $S_4$

9db30420 1fb6e9de a7be7bef d273a298 4af7b7db 64ad8c57 85510443 fa020ed1
7e287aff e60f6bfe 095f35a1 79ebf120 fd05d943 6497b7b1 f3614f63 241e4ad4
28147f5f 4fa2b8cd c9430040 0cc32200 fd30b030 c0a5374f 1d2d00d9 24147b15
ee4d111a 05a554b7 a92d8c57 0137f5ce 85f9d9e5 0137f5ce 85f9d9e5 0137f5ce
80530100 0b73953e 7f72d707 0d0f8f3d 7f8f87b3 0e1b2ee4 9f0a5c38 0f0443d3 60e66dc6 60543a49
5727c148 2be98a1d 8ab4177c 50d218e4 f8548425 9833be5b 600d457d 2b7c3e3c
282f9350 8334b362 d91d1120 2b6d8d8a 642b1e31 9c305a00 52bc6e68 b103588a
f7baefd5 4142ed9c a4315c11 83323ec5 df6fe463 e1353c51 eee53783
1.2 CAST-256 Notation

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

Let $f_1, f_2, f_3$ 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 “$\beta \leftarrow Q_t(\beta)$” be short-hand notation for the following:

\[
\begin{align*}
C &= C \oplus f_1(D, k_{r_0}^{(i)}, k_{m_0}^{(i)}) \\
B &= B \oplus f_2(C, k_{r_1}^{(i)}, k_{m_1}^{(i)}) \\
A &= A \oplus f_3(B, k_{r_2}^{(i)}, k_{m_2}^{(i)}) \\
D &= D \oplus f_1(A, k_{r_3}^{(i)}, k_{m_3}^{(i)})
\end{align*}
\]

Let “$\beta \leftarrow \overline{Q}_t(\beta)$” be short-hand notation for the following:

\[
\begin{align*}
D &= D \oplus f_1(A, k_{r_3}^{(i)}, k_{m_3}^{(i)}) \\
A &= A \oplus f_3(B, k_{r_2}^{(i)}, k_{m_2}^{(i)}) \\
B &= B \oplus f_2(C, k_{r_1}^{(i)}, k_{m_1}^{(i)}) \\
C &= C \oplus f_1(D, k_{r_0}^{(i)}, k_{m_0}^{(i)})
\end{align*}
\]

($Q(\cdot)$ is called a “forward quad-round” and $\overline{Q}(\cdot)$ is called a “reverse quad-round”.)

Let $k_r^{(i)} = \{k_{r_0}^{(i)}, k_{r_1}^{(i)}, k_{r_2}^{(i)}, k_{r_3}^{(i)}\}$ be the set of rotation keys for the $i^{th}$ quad-round, where $k_{r_j}^{(i)}$ is a 5-bit rotation key for $f_1, f_2$, or $f_3$ (as specified above).

Let $k_m^{(i)} = \{k_{m_0}^{(i)}, k_{m_1}^{(i)}, k_{m_2}^{(i)}, k_{m_3}^{(i)}\}$ be the set of masking keys for the $i^{th}$ quad-round, where $k_{m_j}^{(i)}$ is a 32-bit masking key for $f_1, f_2$, or $f_3$ (as specified above).
CAST-256 Notation (cont’d)

Let $\kappa = (ABCDEFGH)$ be a 256-bit block where $A, B, ..., H$ are each 32 bits in length.

Let “$\kappa \leftarrow \omega_i(\kappa)$” be short-hand notation for the following:

$$
G = G \oplus f_1(H, t_{n_1}^{(i)}, t_{m_1}^{(i)}) \\
F = F \oplus f_2(G, t_{n_2}^{(i)}, t_{m_2}^{(i)}) \\
E = E \oplus f_3(F, t_{n_3}^{(i)}, t_{m_3}^{(i)}) \\
D = D \oplus f_4(E, t_{n_4}^{(i)}, t_{m_4}^{(i)}) \\
C = C \oplus f_5(D, t_{n_5}^{(i)}, t_{m_5}^{(i)}) \\
B = B \oplus f_6(C, t_{n_6}^{(i)}, t_{m_6}^{(i)}) \\
A = A \oplus f_7(B, t_{n_7}^{(i)}, t_{m_7}^{(i)}) \\
H = H \oplus f_8(A, t_{n_8}^{(i)}, t_{m_8}^{(i)})
$$

($\omega(\cdot)$ is called a “forward octave”.

Let “$k^{(i)}_r \leftarrow \kappa$” be short-hand notation for the following:

$$
k_{n_1}^{(i)} = 5\text{LSB}(A), \ k_{n_2}^{(i)} = 5\text{LSB}(C), \ k_{n_3}^{(i)} = 5\text{LSB}(E), \ k_{n_4}^{(i)} = 5\text{LSB}(G)
$$

where $5\text{LSB}(x)$ denotes “the five least significant bits of $x$”.

Let “$k^{(i)}_m \leftarrow \kappa$” be short-hand notation for the following:

$$
k_{m_0}^{(i)} = H, \ k_{m_1}^{(i)} = F, \ k_{m_2}^{(i)} = D, \ k_{m_3}^{(i)} = B
$$
1.3 The CAST-256 Cipher

\[ \beta = 128 \text{ bits of plaintext.} \]

\[ \text{for}(i = 0; i < 6; i++) \]
\[ \beta \leftarrow Q_i(\beta) \]

\[ \text{for}(i = 6; i < 12; i++) \]
\[ \beta \leftarrow Q_i(\beta) \]

128 bits 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 \( k_{r(i)}, k_{m(i)} \) derived from \( K \) are used in reverse order as follows.

\[ \text{for}(i = 0; i < 12; i++) \{ \]
\[ k_{r_{\text{rev}}(i)} = k_{r(11-i)} \]
\[ k_{m_{\text{rev}}(i)} = k_{m(11-i)} \]
\}
1.4 The CAST-256 Key Schedule

Initialization:
\[
\begin{align*}
c_m & = 2^{30} \sqrt{2} = 5A827999_{16} \\
m_m & = 2^{30} \sqrt{3} = 6ED9EBA1_{16} \\
c_r & = 19 \\
m_r & = 17
\end{align*}
\]

\[\text{for}(i = 0; i < 24; i++)\]
\[\text{for}(j = 0; j < 8; j++)\]
\[
\begin{align*}
t_{m_j}^{(i)} & = c_m \\
c_m & = (c_m + m_m) \mod 2^{32} \\
t_{r_j}^{(i)} & = c_r \\
c_r & = (c_r + m_r) \mod 32
\end{align*}
\]

Key Schedule:
\[
\kappa = ABCDEFGH = 256\text{ bits of primary key, } K .
\]

\[\text{for}(i = 0; i < 12; i++)\]
\[
\begin{align*}
\kappa & \leftarrow \omega_{z_i}(\kappa) \\
\kappa & \leftarrow \omega_{z_{i+1}}(\kappa) \\
k_r^{(i)} & \leftarrow \kappa \\
k_m^{(i)} & \leftarrow \kappa
\end{align*}
\]

Note:
\[
\begin{align*}
|K| = 128 & \Rightarrow (E = F = G = H = 0) \\
|K| = 160 & \Rightarrow (F = G = H = 0) \\
|K| = 192 & \Rightarrow (G = H = 0) \\
|K| = 224 & \Rightarrow (H = 0)
\end{align*}
\]
2. Design Rationale

2.1 Overall Structure

The fundamental mechanism for the expansion of a 64-bit block size to a larger block size is the generalization of the basic Feistel network (Schneier and Kelsey [SK96] have referred to the structure used here as an “incomplete” Feistel network). The motivation is as follows. In a traditional Feistel network (such as DES), rather than thinking of the exchange of left and right halves in each round as a “swap”, it may be viewed as a circular right-shift of 32 bits. Such a view allows one to consider a cipher with a block size of $32n$ bits, which uses the same round function as the original cipher but requires $n$ rounds (instead of 2) to input all bits of the block to the round function.

A picture may help to clarify the operation.

![Diagram](image.png)

*Figure 1*
The left-most diagram is the “traditional” Feistel network. If this describes two rounds of DES, then \( L \) and \( R \) are each 32 bits in length and the cipher has a 64-bit block size. Continuing the illustration, the middle diagram describes an extended Feistel 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. In each case, we may think of the number of rounds shown as a basic “unit” (in terms of submitting all input bits to the round function); the actual number of rounds chosen for the full cipher will be some multiple of this “unit” (e.g., for DES, the multiple is 8).

2.2 Decryption Considerations

The disadvantage of the generalized structure given above is that it requires a separate structure for decryption (since data must be left-shifted, rather than right-shifted, in each round in order to go backwards through the rounds). By contrast, with the “traditional” Feistel network decryption and encryption are identical except for a change in the ordering of the round keys so no separate structure is needed. Clearly, in constrained environments (such as hardware or firmware implementations that are very resource-limited) requiring two structures is unattractive.

A simple solution to the above concern 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. This simplifies implementation and operation of the cipher and helps to make its use feasible in resource-limited environments.

2.3 Choice of Round Function

One of the very attractive features of the generalized structure given above is that it enables direct re-use of the round function from the “traditional” Feistel network. Within the class of DES-like ciphers, it is well known that increasing the size of the round function typically involves increasing the size of its component substitution boxes (s-boxes); it is also well known that increasing s-box size is generally difficult. For those ciphers that already employ large s-boxes, size increases can be a monumental task. [As a particular example, doubling the input and output sizes of a carefully-constructed \( 8 \times 32 \) s-box may require a work factor of roughly \( 2^{64} \) steps (more than is necessary to break DES by exhaustive search!), aside from the fact that the resulting s-box grows from 4 Kbytes to more than half a million bytes of memory.] Being able to re-use the original round function is therefore very desirable. The important technical decision, however, is which “traditional” Feistel network round function to use in the generalized network.
The CAST-128 set of round functions has a number of appealing features.

- Firstly, the component bent-function-based s-boxes are designed according to a mathematical procedure which produces substitution boxes with several important cryptographic properties (such as high nonlinearity, low XOR difference distribution table values, good higher-order Strict Avalanche Criterion, and good higher-order (Output) Bit Independence Criterion) [A97b].

- Secondly, the use of both a “masking” key and a “rotation” key ensures that the key entropy is higher than the data entropy in each round (following the recommendation of [RPD97]) and appears to make the construction of iterative statistical attacks such as linear and differential cryptanalysis significantly more difficult (or impossible) [A97b].

- Thirdly, the mixing of operations from different algebraic groups (addition modulo 2 and addition / subtraction modulo $2^{32}$) appears to be effective not only in reducing the probability of the round differential [AM97, O’C98], but in reducing the possibility of higher-order differential attacks as well [MSK98].

- Finally, mixing the order of the group operations (i.e., by varying the order of round functions throughout the cipher, as is done in CAST-128) appears to frustrate the practical construction of iterative characteristics.

In summary, then, the extensive analysis done on the CAST design procedure (including focused attention within several master’s- and doctoral-level theses on symmetric cipher design and analysis) lends confidence to its choice as the round function for this generalized Feistel network.

[See CAST-256: Algorithm Analysis below for a partial list of published work which discusses or analyzes various aspects of the CAST design procedure. For one significant example of unpublished work that has been done on CAST, the Communications Security Establishment, after extensive analysis, has determined and will formally state that the CAST-128 algorithm is suitable for the protection of all levels of Designated information within the Government of Canada. Please see the attached letter dated June 5th, 1998, and note that “CAST5” is the name used for “CAST-128” when specific key lengths are explicitly intended (see [A97c], Section 2.5).]
2.4 Number of Rounds

Given that the basic unit (see “Overall Structure” above) in DES is a “double round” and that a multiple of 8 is used to give the full 16-round cipher, it is reasonable to conclude that a 128-bit block size, with a “quad-round” as the basic unit, should consist of at least 32 rounds for the full cipher. It is important to note, however, that a cipher being constructed as a candidate for AES consideration must support not only twice the block size of CAST-128, but twice the key size as well. A deeper security analysis (see attached document, CAST-256: Algorithm Analysis) suggests that 48 rounds (i.e., 12 “quad rounds”) provides security protection commensurate with the parameters of the desired cipher.

2.5 Key Schedule

Key scheduling (deriving a set of round keys from an initial key) is an extremely important aspect of cipher design since sub-optimal key schedules can lead to exploitable weaknesses in the cipher (including weak keys, equivalent keys, complementation properties, and susceptibility to related-key attacks), and overly-complicated key schedules can lead to prohibitively-long set-up times (limiting the use of the cipher in some environments).

The design philosophy chosen for the CAST-256 key schedule is identical to that chosen for the CAST-256 cipher itself: the key schedule essentially describes a generalized Feistel network with a 256-bit block size. A simple (but fixed) set of round keys is used to key this network and the CAST-256 initial key is used as the plaintext input. Some of the output bits of selected rounds during this “encryption” define the actual round keys for the CAST-256 cipher. Important features of this key scheduling approach include the following.

- The inherent strength of the generalized Feistel network is used in the key schedule to create round keys, increasing confidence that the set of key values (comprised of the generated round keys and the CAST-256 initial key) will appear to be pair-wise independent to any statistical analysis.

- If an attack can be mounted that derives four or more full round keys (i.e., full masking keys and the corresponding rotation keys) from the CAST-256 cipher, it still appears to require a computational effort of at least \(2^{256 - (4 \times 32) - (4 \times 5)} = 2^{108}\) guesses to derive the CAST-256 initial key from this information.
Since the key schedule describes a generalized Feistel network, it is extremely unlikely that key collisions can occur. The key schedule defines a cipher with a fixed key (i.e., a permutation over the input space) so for two different CAST-256 initial keys to produce identical sets of round keys, the different cipher inputs would have to map to round function outputs (in every relevant round) that differed only in the 108 bits not used to produce round key bits. The probability of this occurring in each octave that produces round keys is $2^{108}/2^{256} = 2^{-148}$, so the probability that this occurs over the full set of round keys is $2^{-148+12} = 2^{-1776}$ (essentially zero, since there are only $2^{256}$ possible initial keys).

The key scheduling operation requires the equivalent of four CAST-256 encryption operations to produce a full set of round keys. This ratio is not prohibitive for most environments and compares favorably with many current implementations of DES.

The key schedule chosen for CAST-256 appears to have a number of desirable cryptographic features and takes into account much of the research into key schedule design and analysis over the past two decades (see, for example, [A94] and the references included in [A97]).

### 2.6 Conclusions

A number of alternatives exist for doubling the block size of a cipher from 64 bits to 128 bits, including the following.

- **Feistel network.** In such a design, the round function of the Feistel network is the original 64-bit cipher, which may itself be a Feistel network (this is a simple extension of ideas presented in, for example, [LR88, L96]).

- **Substitution-Permutation (SP) network** [F73]. In such a design, two parallel implementations of the original cipher are used as the substitution layers; these are interspersed with an extended permutation layer (i.e., a permutation which is the width of the desired block size).

- **“Fenced” Construction** [R96]. In such a design, two parallel implementations of the original cipher are surrounded by specially-constructed mixing operations, which in turn are surrounded by a layer of substitution boxes.
However, it was felt that all the alternatives considered had one or more drawbacks which made them somewhat less attractive as AES submission candidates. For example, the Feistel network suffers significant security degradation if one or two rounds may be “peeled off” by some attack (not an uncommon situation) since the entire outer network would likely consist of only four or six rounds (for performance reasons). The SP network may be subject to poor encryption / decryption performance since even two substitution layers with a permutation layer in between (the minimum possible configuration) halves the speed of the original cipher; a larger number of layers decreases performance significantly beyond this. Finally, the Fenced construction has non-trivial design and implementation impacts with the need for solid theoretical justification for the particular mixing operations used and the need for sufficient processing time and memory for the pseudo-random generation and storage of the necessary s-boxes.

The approach taken in this proposal to achieve block size doubling (i.e., the use of a generalized Feistel network) appears to be the simplest and most elegant of the various alternatives. It has none of the drawbacks listed above, is straightforward to understand and to analyze, and builds on the confidence gained from the extensive literature on ciphers based on Feistel networks. Furthermore, it allows unmodified re-use of a round function with a number of attractive cryptographic features, and suggests an intuitive architecture for the associated key scheduling algorithm.

We conclude that the rationale for CAST-256 is solid, resting on firm theoretical results and immediately appealing, defensible, concepts for every aspect of the cipher design. The resulting algorithm has good performance, reasonable code and memory size, and high security (according to all analysis conducted to date); it thus appears to meet all the requirements for an AES submission candidate.
3. Bit Naming / Numbering Convention Provided

True (needed only in section 1.1 CAST-128 Notation above, where most- to least-significant bytes of a 32-bit word are specified).

4. No Parity Bits Specified in the Key Definition

True.

5. References


[AM97] C. Adams and S. Mister, Preliminary experimental results concerning the mixing of operations from different algebraic groups and the contents of the resulting XOR difference distribution table (unpublished).


[O‘C98] L. O‘Connor, Preliminary analytical results concerning the mixing of operations from different algebraic groups and the maximum value of the resulting XOR difference distribution table (unpublished).


05 June 1998

Mr. Brian O’Higgins
Executive Vice President and
Chief Technology Officer
Entrust Technologies
750 Heron Road
Suite 800
Ottawa, Ontario
K1V 1A7

Dear Mr. O’Higgins,

I am very pleased to advise you that CSE has completed its evaluation of the CAST5 algorithm (80 and 128 bit versions). We have determined that CAST5 is suitable for the protection of all levels of Designated information within the GOC. A formal statement of this approval will be promulgated to Government of Canada departments and agencies in the very near future.

On behalf of the Communications Security Establishment please accept my congratulations.

David McKerrow
Communications Security Establishment
Director General Information Technology Security
CAST-256
Computational Efficiency

1. Efficiency Estimates for the NIST AES Analysis Platform

1.1 Platform Description

IBM-compatible PC, with an Intel Pentium Pro Processor, 200MHz clock speed, 64MB RAM, running Windows95.

1.2 Speed Estimates (in clock cycles)

<table>
<thead>
<tr>
<th>Operation</th>
<th>128/128</th>
<th>192/128</th>
<th>256/128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt one data block:</td>
<td>1790</td>
<td>1790</td>
<td>1790</td>
</tr>
<tr>
<td>Decrypt one data block:</td>
<td>1790</td>
<td>1790</td>
<td>1790</td>
</tr>
<tr>
<td>Key setup:</td>
<td>9090</td>
<td>9090</td>
<td>9090</td>
</tr>
<tr>
<td>Algorithm setup:</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Key change:</td>
<td>9090</td>
<td>9090</td>
<td>9090</td>
</tr>
</tbody>
</table>

1.3 Tradeoffs Between Speed and Memory

For environments in which memory is not a scarce resource, s-boxes $S_1$ and $S_2$ can be combined into three $16\times32$ s-boxes (one corresponding to $S_1\oplus S_2$, one corresponding to $S_1-S_2$, and one corresponding to $S_1+S_2$, 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.
2. Efficiency Estimates for 8-Bit Processors

2.1 Platform Description

Motorola 6811 microprocessor, 2MHz clock speed, assembly language implementation.

2.2 Speed Estimates (in clock cycles)

<table>
<thead>
<tr>
<th>Operation</th>
<th>128/128</th>
<th>192/128</th>
<th>256/128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt one data block:</td>
<td>26000</td>
<td>26000</td>
<td>26000</td>
</tr>
<tr>
<td>Decrypt one data block:</td>
<td>26000</td>
<td>26000</td>
<td>26000</td>
</tr>
<tr>
<td>Key setup:</td>
<td>110000</td>
<td>110000</td>
<td>110000</td>
</tr>
<tr>
<td>Algorithm setup:</td>
<td>0 ms</td>
<td>0 ms</td>
<td>0 ms</td>
</tr>
<tr>
<td>Key change:</td>
<td>110000</td>
<td>110000</td>
<td>110000</td>
</tr>
</tbody>
</table>

2.3 Tradeoffs Between Speed and Memory

None known.
3. Efficiency Estimates for Other Platforms

3.1 Platform Description

IBM-compatible PC, with an Intel Pentium II Processor, 300MHz clock speed, 128MB RAM, running Windows NT 4.0, assembly language implementation.

3.2 Speed Estimates (in clock cycles)

<table>
<thead>
<tr>
<th>Operation</th>
<th>128/128</th>
<th>192/128</th>
<th>256/128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt one data block:</td>
<td>815</td>
<td>815</td>
<td>815</td>
</tr>
<tr>
<td>Decrypt one data block:</td>
<td>815</td>
<td>815</td>
<td>815</td>
</tr>
<tr>
<td>Key setup:</td>
<td>4130</td>
<td>4130</td>
<td>4130</td>
</tr>
<tr>
<td>Algorithm setup:</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Key change:</td>
<td>4130</td>
<td>4130</td>
<td>4130</td>
</tr>
</tbody>
</table>

3.3 Tradeoffs Between Speed and Memory

For environments in which memory is not a scarce resource, s-boxes $S_1$ and $S_2$ can be combined into three $16\times32$ s-boxes (one corresponding to $S_1\oplus S_2$, one corresponding to $S_1-S_2$, and one corresponding to $S_1+S_2$, 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.
4. Efficiency Estimates for Other Platforms

4.1 Platform Description

Sun UltraSparc 1, 167MHz clock speed, 124MB RAM, running Solaris 2.5.

4.2 Speed Estimates (in clock cycles)

<table>
<thead>
<tr>
<th>Operation</th>
<th>128/128</th>
<th>192/128</th>
<th>256/128</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypt one data block:</td>
<td>1180</td>
<td>1180</td>
<td>1180</td>
</tr>
<tr>
<td>Decrypt one data block:</td>
<td>1180</td>
<td>1180</td>
<td>1180</td>
</tr>
<tr>
<td>Key setup:</td>
<td>5830</td>
<td>5830</td>
<td>5830</td>
</tr>
<tr>
<td>Algorithm setup:</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Key change:</td>
<td>5830</td>
<td>5830</td>
<td>5830</td>
</tr>
</tbody>
</table>

4.3 Tradeoffs Between Speed and Memory

For environments in which memory is not a scarce resource, s-boxes $S_1$ and $S_2$ can be combined into three $16 \times 32$ s-boxes (one corresponding to $S_1 \oplus S_2$, one corresponding to $S_1 - S_2$, and one corresponding to $S_1 + S_2$, 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.
5. General Efficiency Comments

As will be noted in the tables given above, CAST-256 has the following features:

- it requires no algorithm setup time (e.g., there is no need to generate s-boxes or other tables, and no need to pre-compute values);
- decryption performance is identical to encryption performance;
- key change time is identical to key setup time;
- there is no penalty for key size differences (i.e., encryption / decryption performance and key setup performance are unaffected by whether the primary key is 128 bits, 256 bits, or a value in between).
CAST-256
Algorithm Analysis

1. Analysis With Respect to Known Attacks

The classical attacks on ciphers are as follows: *ciphertext only; known plaintext; and chosen plaintext*. The advent of public-key cryptography added utility to the concept of a *chosen ciphertext* attack, but this appears to be of little added value in the analysis of symmetric ciphers. Research in the past decade or so has also introduced the notions of *chosen key* and *related key* attacks, which have enjoyed some success in the cryptanalysis of specific symmetric ciphers. Within the iterated symmetric ciphers (the class of algorithms to which CAST-256 belongs), the techniques known as *linear cryptanalysis* and *differential cryptanalysis* (along with their combinations and higher-orders) currently represent the most general and powerful instances of *known plaintext* and *chosen plaintext* attacks, respectively.

This section of the submission package examines the CAST-256 algorithm with respect to the families of cryptanalytic attack listed above.

1.1 Ciphertext Only Attack

No techniques are currently known that will allow an attacker to infer or derive information about the plaintext, the primary key, or any subset of round keys from any collection of ciphertext blocks. The one (unavoidable) exception to this is the technique applicable to all $n$-bit-block ciphers when used in Cipher-Block-Chaining (CBC) mode: once $2^{n/2}$ blocks have been encrypted, with probability roughly $\frac{1}{2}$ (rapidly increasing as more blocks are encrypted) an XOR relationship between a particular pair of plaintexts will be known.

1.2 Known Plaintext Attack: Linear Cryptanalysis

Linear cryptanalysis [M94] attempts to exploit any high-probability occurrences of linear expressions of input, output, and round key bits in the round function of an iterated cipher. It has been approximated [M94] that the best linear expression for $r$-rounds of a cipher has a probability of being satisfied that is bounded as follows:
\[ |p_L - \frac{1}{2}| \leq 2^{\alpha-1} \cdot |p_\beta - \frac{1}{2}|^{\alpha} \]

where \( p_L \) represents the probability that the linear expression holds, \( p_\beta \) represents the probability of the best linear approximation, and \( \alpha \) represents the number of s-boxes involved in that linear approximation. This expression is based on the assumption of independent round keys such that the linear approximations of the s-boxes are independent. In an analogous way to “differentials” and “characteristics” in differential cryptanalysis, provable immunity in linear cryptanalysis relies on bounding the likelihood of an overall linear expression (sometimes referred to as the “linear hull”) rather than any particular linear “characteristic”. However, for many ciphers (including CAST-256) this is a difficult analytical task. What are typically considered, therefore, are the building blocks of an overall linear expression: the sequence of approximations of the round functions which result in the overall linear expression.

A basic linear attack typically uses a sequence of linear approximations of the rounds to create an overall linear expression involving subsets of plaintext and ciphertext bits. From this it is possible to derive the equivalent of one key bit represented as the XOR of a number of round key bits. In this case, it is shown [M94] that the number of known plaintexts required is approximately

\[ N_L = |p_L - \frac{1}{2}|^{-2} \]

It can be shown that the best linear approximation has a probability given by

\[ |p_\beta - \frac{1}{2}| = \frac{2^{m-1} - NL_{\text{min}}}{2^m} \]

where \( m \) is the number of input bits to the s-box and \( NL_{\text{min}} \) is the nonlinearity of the s-box [LHT97]. For the s-boxes of CAST-256, \( m = 8 \) and \( NL_{\text{min}} = 74 \). Furthermore, for the CAST-256 cipher, the best linear approximation appears to involve 4 s-boxes every 4 rounds such that the linear approximation of the round function for every 4th round involves no output bits. That is, the linear expression used is \( X_{i_1} \oplus X_{i_2} \oplus \ldots \oplus X_{i_4} \), where \( X_{i_j} \) represents an input bit to the s-box. Hence, for an \( r \)-round linear approximation, \( \alpha = r \). The number of known plaintexts required for a 48-round linear approximation of CAST-256, then, is approximately \( 2^{122} \). Note that this is almost equal to the total number of plaintexts available \((2^{128})\) and argues against the practicality of a linear attack on this cipher.

Furthermore, Youssef, et al, have proposed [YCT97] that a more accurate bound on the number of plaintexts required for linear cryptanalysis of a CAST cipher can be obtained by considering the combination of s-boxes in the round function rather than the individual s-boxes. In particular, they compute the value for \( NL_S \), the nonlinearity of the composite
32×32 s-box when the individual 8×32 s-boxes are combined using XOR. Using this in place of $NL_{\text{min}}$ in the equations above and setting $m = 32$ and $\alpha = \frac{1}{2}$ (since an $r$-round linear approximation must involve at least as many 32×32 s-boxes as $r/2$ iterations of the best 2-round approximation) yields a number of known plaintexts required for a 48-round linear approximation at more than $2^{174}$ (far beyond the number of plaintexts available). Note that experimental evidence suggests that combining s-boxes using mixed operations may increase the nonlinearity of the composite s-box even further.

It therefore appears that CAST-256 is immune to a linear cryptanalysis attack.

1.3 Chosen Plaintext Attack: Differential Cryptanalysis

Differential cryptanalysis [BS93] attempts to exploit any high-probability output differences resulting from particular input differences in the round function of an iterated cipher. A block cipher can be proved to be resistant to differential cryptanalysis if it can be shown that no high-probability differentials [LMM91] exist, where an $i$-round differential is defined to be the XOR of two outputs after $i$ rounds, where the outputs correspond to two plaintexts with a given XOR.

In a good cipher the probability of all differentials should approach $2^{-N}$, where $N$ is the block size. Strictly speaking, differential cryptanalysis requires only the existence of a highly-probable differential to succeed. However, differentials can be viewed to be comprised of a number of possible characteristics, where a characteristic specifies the exact sequence of input and output XORs for each round to achieve the overall differential input and output XOR.

It is typically difficult to derive the probability of any particular differential and, in practice, it would be hard for a cryptanalyst to determine the existence of a highly-probable differential without searching for highly-probable characteristics. Although it is often the case that an upper bound on the probability of a differential cannot be stated for a particular cipher (that is, resistance to a differential cryptanalytic attack cannot be proved), the probabilities of the most likely characteristics can be determined. These probabilities can then be used as a measure of the cipher’s resistance to differential cryptanalysis.

As is common in the literature, the analysis here is based on the assumption that all round keys are independent (although this assumption is not always necessary; see [C97]) and that the occurrence of output XORs given particular input XORs is independent for different rounds. Under such conditions, the probability of an $r$-round characteristic is given by

$$p_{\Omega_r} = \prod_{i=1}^{r} p_i$$
where \( p_i \) represents the probability of the output XOR given the input XOR in round \( i \).

The best characteristics that can be constructed are typically iterative in nature. For the CAST-256 cipher with \( R \) rounds, the following appears to be the best possible \( r \)-round characteristic, where \( r \) is a multiple of 4. (Note that the notation \((W,X,Y,Z)\) represents XOR vectors for the four 32-bit sub-blocks in a CAST-256 round function input.)

\[
\begin{align*}
(0,0,0,\Delta) & \quad \text{[input XOR to round 1]} \\
0 & \leftarrow \Delta \text{ with probability } p \quad \text{[round 1]} \\
0 & \leftarrow 0 \text{ with probability } 1 \quad \text{[round 2]} \\
0 & \leftarrow 0 \text{ with probability } 1 \quad \text{[round 3]} \\
0 & \leftarrow 0 \text{ with probability } 1 \quad \text{[round 4]} \\
& \quad \text{... repeat up to } R/2 \text{ rounds} \\
(0,\Delta,0,0), \text{ or some variation} & \quad \text{[input XOR to round } (R/2 + 1)\text{]} \\
0 & \leftarrow 0 \text{ with probability } 1 \quad \text{[round } (R/2 + 1)\text{]} \\
0 & \leftarrow 0 \text{ with probability } 1 \quad \text{[round } (R/2 + 2)\text{]} \\
0 & \leftarrow \Delta \text{ with probability } p \quad \text{[round } (R/2 + 3)\text{]} \\
0 & \leftarrow 0 \text{ with probability } 1 \quad \text{[round } (R/2 + 4)\text{]} \\
& \quad \text{... repeat up to } r \text{ rounds for } r\text{-round char.}
\end{align*}
\]

The input XOR to round \((R/2+1)\) will be a vector in which one of the sub-blocks is non-zero and the other three sub-blocks are zero (the precise variation which applies for a given cipher depends upon the value of \( R \)). Without loss of generality, the example \((0,\Delta,0,0)\) is shown above.

As per the analysis and rationale given in [LHT97], the input-output XOR pair for a simplified CAST round function (i.e., one which does not include the key-dependent rotation, and for which the only s-box combining operation used is XOR) can be assumed to have a probability of \( p \leq 2^{-14} \). This is based on the fact that all four s-boxes in the CAST round function are injective and the format of the XOR pair has the output XOR being equal to 0. This leads to the conclusion that the best \( r \)-round iterated characteristic as shown above has a probability given by

\[
p_{\alpha_i} \leq (2^{-14})^{r/4}
\]

In particular, a 40-round characteristic must have a probability less than or equal to \( 2^{-140} \) according to the assumptions of the analysis. This implies that the number of chosen plaintexts required for this attack would be greater than \( 2^{140} \) for the 48-round cipher (substantially greater than the number of plaintexts available for a 128-bit block size).

It therefore appears that CAST-256 is immune to a differential cryptanalysis attack.
1.4 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.

1.5 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.

1.6 Enhancements to the Above Statistical Attacks: Combinations and Higher-Orders

The analysis given above for both linear and differential cryptanalysis applies to a greatly simplified version of the CAST-256 cipher. The actual cipher, which includes key-dependent rotation and mixed operations in the round function (both for data masking and for s-box combination), appears to be much more difficult/impossible to attack using the methods as described in [M94] and [BS93] (see [A97] for some discussion of this). In particular, experiments in which two CAST-256 s-boxes are combined using addition or subtraction modulo $2^{32}$ show that the maximum value in the XOR difference distribution table is approximately 10% of the maximum that occurs when the s-boxes are combined using XOR. Experiments on combinations of three CAST-256 s-boxes are on-going, but thus far show similar results. This lends confidence that combinations of four s-boxes using mixed operations (as is done in the CAST-256 round function) are effective in increasing resistance to differential cryptanalysis.

The above experimental work [AM97] is supported by a new analytical result [O’C98], which shows that for a random $n$-bit permutation, the probability that the maximum entry in a differential table based on XOR differences is greater than a bound $B_n$ approaches 1 as $n$ grows, whereas the probability that the maximum entry in a table based on non-XOR differences (e.g., modular addition or multiplication) is greater than that same bound approaches 0. Furthermore, the bound is accurate for the 8-bit case. Thus, although the details of the analyzed structure differ slightly from the internals of the CAST-256 round
function as used in the above experiments, the conclusion is the same: using operations from different algebraic groups appears to be helpful in increasing resistance to differential cryptanalysis (by lowering the differential probability of a single round).

1.6.1 Combination Attacks

CAST-256 appears to be immune to both linear and differential cryptanalysis (requiring more plaintext than is available from the 128-bit block size) and appears to be immune to both chosen and related key attacks (due to the absence of exploitable statistical correlations among its generated keys). Given this, it seems highly unlikely that various combination attacks (such as linear-differential, or differential-related-key) can have any measure of success.

It therefore appears that this cipher is immune to the combination attacks currently known in the literature.

1.6.2 Higher-Order Attacks

The concept of higher-order differentials has been introduced [L94, K95] and used to successfully cryptanalyze ciphers proved secure against ordinary differential cryptanalysis [JK97]. A simplified version of the CAST-128 cipher (one which uses XOR for all operations in the round function) has been examined with respect to the higher-order differential attack [MSK98]. It has been shown that this attack is successful up to 5 rounds, but cannot be extended to higher numbers of rounds. Furthermore, the introduction of the key-dependent rotation operation is effective in increasing the computational complexity of this attack. Finally, the use of operations from different algebraic groups “makes the degree too high to cryptanalyze by the higher-order differential attack” [MSK98], so that the attack cannot even be mounted on a 5-round version of the cipher.

It therefore appears that CAST-256 (which has 48 rounds and uses the CAST-128 round functions) is immune to a higher-order differential attack.
2. Statements Regarding Properties of Keys

This section provides statements regarding the following properties of keys with respect to CAST-256: *weak keys*, *semi-weak keys*, *fixed points of a key*, *equivalent keys*, and *restrictions on key selection*. It also includes a statement on *complementation properties* since this is sometimes related to the way that round keys are used within a DES-like cipher.

2.1 Weak Keys

None known. In the CAST-256 cipher, all keys appear to be of equivalent strength and are usable for double encryption (i.e., no key appears to be its own inverse).

2.2 Semi-Weak Keys

None known. In the CAST-256 cipher, there appear to be no pairs of keys which cannot be used for double encryption (i.e., there do not appear to be pairs of keys $k_i$ and $k_j$ such that $k_j$ is the inverse of $k_i$).

2.3 Fixed Points of a key $K$

None known. From all evidence available thus far in the open literature, fixed points have only been easily found (i.e., requiring a level of effort for an $n$-bit block cipher of roughly $2^{n/2}$ operations rather than $2^n$ operations) in DES-like ciphers for weak and semi-weak keys. It therefore appears that CAST-256 has no easily-found fixed points for any key.

2.4 Equivalent Keys

None known. The key schedule defines a cipher with a fixed key (i.e., a permutation over the input space) so for two different CAST-256 initial keys to produce identical sets of round keys, the different cipher inputs would have to map to round function outputs (in every relevant round) that differed only in the 108 bits not used to produce round key bits. The probability of this occurring in each octave that produces round keys is $2^{108} / 2^{256} = 2^{-148}$, so the probability that this occurs over the full set of round keys is $2^{148*12} = 2^{-1776}$ (essentially zero, since there are only $2^{256}$ possible initial keys).
2.5 Restrictions on Key Selection

None known. The key scheduling algorithm defines a symmetric block cipher with a fixed key where the CAST-256 primary key is used as the plaintext input. Because in this symmetric block cipher there are no restrictions on the input space (i.e., any plaintext can be encrypted), it follows that no restrictions are placed upon selection of CAST-256 primary keys.

2.6 Complementation Properties

None known. There appear to be no complementations of combinations of plaintext, key, and ciphertext that lead to identities. This is due to the complexity of the key scheduling operation (so that complementing the primary key leads to random-looking changes to all round keys) and also to the use of multiple operations to combine data, key, and s-boxes within the round functions (XOR, rotation, and addition and subtraction modulo $2^{32}$).
3. Statement Regarding Trap-Doors

None known. There are several reasons to feel confident that there are no trap-doors in this cipher.

- CAST-256 uses the four round function s-boxes in CAST-128. The design criteria and construction procedure for these s-boxes have been published [A97, MA96] and the specific s-boxes themselves have been examined by a number of researchers.

- CAST-256 uses the three round functions in CAST-128. The design criteria for these round functions have been published [A97] and the specific round functions themselves have been examined by a number of researchers. Furthermore, the complexity introduced by the mixed operations in the round functions would seem to make it difficult to insert a trap-door of any kind.

- CAST-256 uses 48 rounds. Inserting a non-obvious trap-door that will carry through 48 rounds of the cipher would seem to be a formidable task.

- CAST-256 uses a significantly more complex key scheduling algorithm than DES. A trap-door in the final round that allows the attacker (i.e., the one knowing this trap-door) to recover information about the final round key will be of little help in deriving either other round keys or the primary key. This contrasts with DES in which knowledge of any round key gives knowledge of the primary key with only a brute-force search over 8 bits of key.
4. Publications Discussing or Analyzing Aspects of the CAST Design Procedure


S. Mister and C. Adams, “Practical S-Box Design”, Workshop on Selected Areas in Cryptography, SAC ’96, Workshop Record, 1996, pp.61-76.


[Please note that a number of the above papers are available at the following location: http://adonis.ee.queensu.ca:8000/cast/ ]
5. References


[AM97] C. Adams and S. Mister, Preliminary experimental results concerning the mixing of operations from different algebraic groups and the contents of the resulting XOR difference distribution table (unpublished).


[O'C98] L. O’Connor, Preliminary analytical results concerning the mixing of operations from different algebraic groups and the maximum value of the resulting XOR difference distribution table (unpublished).