C2 Block Cipher Specification

Intel Corporation

International Business Machines Corporation

Matsushita Electric Industrial Co., Ltd.

Toshiba Corporation

Revision 1.0

January 17, 2003

Notice

Intellectual Property

Implementation of this specification requires a license from the 4C Entity, LLC.

Contact Information

The URL for the 4C Entity, LLC web site is http://www.4Centity.com.

Introduction

The Cryptomeria Cipher (C2) is a Feistel network-based block cipher designed for use in the area of digital entertainment content protection. The cipher was designed for cryptographic robustness, efficiency when implemented in software, and small size when implemented in hardware. The C2 cipher has the following basic characteristics:

Input Block Size	64 bits
Output Block Size	64 bits
Input Key Size	56 bits
Number of Rounds	10

This document contains a description of the cipher algorithm, provided as code written in the C programming language. Four functions are provided, which define the cipher's four basic modes of operation, as follows:

- C2_E: Encryption in Electronic Codebook (ECB) Mode, as defined in ISO 8372 or ISO/IEC 10116
- C2_D: Decryption in ECB Mode
- C2_ECBC: Encryption in Converted Cipher Block Chaining (C-CBC) Mode
- C2_DCBC: Decryption in C-CBC Mode

Unless explicitly noted otherwise, the functions assume big-endian ordering for multiple-byte values, meaning that byte 0 is the most significant byte.

Prologue

```
/*
This source assumes a Big Endian machine (most significant byte
first), where the "long" is 32 bits:
*/

typedef unsigned long WORD32;
typedef unsigned char BYTE;

/*
Logical left rotate macros:
*/
#define lrot8(x,n)    (((x)<<(n))|((x)>>(8-(n))))
#define lrot32(x,n)   (((WORD32)(x)<<(n))|((WORD32)(x)>>(32-(n))))

/*
The secret constant is available under license from the 4C Entity, LLC.
*/
extern const BYTE SecretConstant[256];

/*
The cipher has 10 rounds:
*/
#define MaxRound 10

/*
F is the Feistel round function:
*/
static WORD32 F(WORD32 data, WORD32 key)
{
       WORD32 t;
       BYTE v[4], u;

       /* Key Insersion */
       t = data+key;

       /* Secret Constant */
       v[3] = (BYTE)((t>>24)&0xff);
       v[2] = (BYTE)((t>>16)&0xff);
       v[1] = (BYTE)((t>> 8)&0xff);
       v[0] = SecretConstant[t&0xff];
       u = v[0]^0x65;      v[1] ^= lrot8(u,1);
       u = v[0]^0x2b;      v[2] ^= lrot8(u,5);
       u = v[0]^0xc9;      v[3] ^= lrot8(u,2);

       /* Rotate */
       t = ((WORD32)v[3]<<24)|((WORD32)v[2]<<16)|((WORD32)v[1]<<8)|(WORD32)v[0];
       t ^= lrot32(t,9) ^ lrot32(t,22);

       return t;
}
```

Encryption in ECB Mode

```
/*
C2 encryption in ECB (Electronic Code Book) mode:
*/
void C2_E(BYTE key[7], WORD32 *inout)
{
	WORD32		L, R, t;
	WORD32		ktmpa, ktmpb, ktmpc, ktmpd;
	WORD32		sk[MaxRound];
	int		round;

	/* Input Conversion */
	L = inout[0];
	R = inout[1];

	/* Key Schedule Generation */
	ktmpa = ((WORD32)key[0]<<16)|((WORD32)key[1]<<8)|(WORD32)key[2];
	ktmpb =
((WORD32)key[3]<<24)|((WORD32)key[4]<<16)|((WORD32)key[5]<<8)|(WORD32)key[6];

	for(round=0; round<MaxRound; round++) {
		ktmpa &= 0x00ffffff;
		sk[round] = ktmpb + ((WORD32)SecretConstant[(ktmpa&0xff)^round]<<4);
		/* This is a 56 bit rotate: */
		ktmpc = ktmpb >> (32-17);
		ktmpd = ktmpa >> (24-17);
		ktmpa = (ktmpa << 17)|ktmpc;
		ktmpb = (ktmpb << 17)|ktmpd;
	}

	for(round=0; round<MaxRound; round++) {
		/* Feistel net: */
		L += F(R, sk[round]);
		t = L;  L = R;  R = t;     // swap
	}
	t = L;  L = R;  R = t;     // swap cancel

	/* Output */
	*inout++ = L;
	*inout++ = R;

	return;
}
```

Decryption in ECB Mode

```
/*
C2 decryption in ECB (Electronic Code Book) mode:
*/
void C2_D(BYTE key[7], WORD32 *inout)
{
	WORD32		L, R, t ;
	WORD32		ktmpa, ktmpb, ktmpc, ktmpd;
	WORD32		sk[MaxRound];
	int		round;

	/* Input Conversion */
	L = inout[0];
	R = inout[1];

	/* Key Generation */
	ktmpa = ((WORD32)key[0]<<16)|((WORD32)key[1]<<8)|(WORD32)key[2];
	ktmpb =
((WORD32)key[3]<<24)|((WORD32)key[4]<<16)|((WORD32)key[5]<<8)|(WORD32)key[6];

	for(round=0; round<MaxRound; round++) {
		ktmpa &= 0x00ffffff;
		sk[round] = ktmpb + ((WORD32)SecretConstant[(ktmpa&0xff)^round]<<4);
		/* 56 bit left rotate */
		ktmpc = ktmpb >> (32-17);
		ktmpd = ktmpa >> (24-17);
		ktmpa = (ktmpa << 17)|ktmpc;
		ktmpb = (ktmpb << 17)|ktmpd;
	}

	for(round=MaxRound-1; round>=0; round--) {
		/* Feistel net */
		L -= F(R, sk[round]);
		t = L;  L = R;  R = t;
	}

	t = L;  L = R;  R = t;	// swap cancel

	/* Output */
	*inout++ = L;
	*inout++ = R;

	return;
}
```

Encryption in C-CBC Mode

```
/*
C2 encryption in C-CBC (Converted Cipher Block Chaining) mode:
*/
void C2_ECBC(const BYTE key[7], WORD32 *inout, int length)
{
	WORD32  L, R, t;
	WORD32  ktmpa, ktmpb, ktmpc, ktmpd;
	WORD32  sk[MaxRound];
	BYTE    inkey[7];
	int     keyRound = MaxRound;   /* first time through, then becomes 2 */
	int     round, i;              /* loop variables */

	memcpy(inkey, key, 7);

	for (i=0; i < length; i+=8) {
		/* Input Conversion */
		L = inout[0];
		R = inout[1];

		/* Key Schedule Generation */
		ktmpa = ((WORD32)inkey[0]<<16)|((WORD32)inkey[1]<<8)|(WORD32)inkey[2];
		ktmpb =
((WORD32)inkey[3]<<24)|((WORD32)inkey[4]<<16)|((WORD32)inkey[5]<<8)|(WORD32)inkey[6];

		for(round=0; round<keyRound; round++) {
			ktmpa &= 0x00ffffff;
			sk[round] = ktmpb + ((WORD32)SecretConstant[(ktmpa&0xff)^round]<<4);
			/* This is a 56 bit rotate: */
			ktmpc = ktmpb >> (32-17);
			ktmpd = ktmpa >> (24-17);
			ktmpa = (ktmpa << 17)|ktmpc;
			ktmpb = (ktmpb << 17)|ktmpd;
		}

		for(round=0; round<MaxRound; round++) {
			/* Feistel net: */
			L += F(R, sk[round % keyRound]);
			if (round == 4) {
				/* key chaining */
				inkey[0] = key[0] ^ (R >> 16);
				inkey[1] = key[1] ^ (R >> 8);
				inkey[2] = key[2] ^ R;
				inkey[3] = key[3] ^ (L >> 24);
				inkey[4] = key[4] ^ (L >> 16);
				inkey[5] = key[5] ^ (L >> 8);
				inkey[6] = key[6] ^ L;
			}
			t = L;  L = R;  R = t;  // swap
		}
		t = L;  L = R;  R = t;  // swap cancel


		/* Output */
		*inout++ = L;
		*inout++ = R;

		/* Subsequent blocks after the first use a truncated key schedule: */
		keyRound = 2;
	}

	return;
}
```

Decryption in C-CBC Mode

```
/*
C2 decryption in C-CBC (Converted Cipher Block Chaining) mode:
*/
void C2_DCBC(const BYTE key[7], WORD32 *inout, int length)
{
	WORD32  L, R, t;
	WORD32  ktmpa, ktmpb, ktmpc, ktmpd;
	WORD32  sk[MaxRound];
	BYTE    inkey[7];
	int     keyRound = MaxRound;
	int     i, round;

	memcpy(inkey, key, 7);

	for (i=0; i < length; i+=8) {

		/* Input Conversion */
		L = inout[0];
		R = inout[1];

		/* Key Generation */
		ktmpa = ((WORD32)inkey[0]<<16)|((WORD32)inkey[1]<<8)|(WORD32)inkey[2];
		ktmpb =
((WORD32)inkey[3]<<24)|((WORD32)inkey[4]<<16)|((WORD32)inkey[5]<<8)|(WORD32)inkey[6];

		for(round=0; round<keyRound; round++) {
			ktmpa &= 0x00ffffff;
			sk[round] = ktmpb + ((WORD32)SecretConstant[(ktmpa&0xff)^round]<<4);
			/* 56 bit left rotate */
			ktmpc = ktmpb >> (32-17);
			ktmpd = ktmpa >> (24-17);
			ktmpa = (ktmpa << 17)|ktmpc;
			ktmpb = (ktmpb << 17)|ktmpd;
		}

		for(round=MaxRound-1; round>=0; round--) {
			/* Feistel net */
			L -= F(R, sk[round%keyRound]);
			t = L;  L = R;  R = t;
			if (round == 5) {
				/* key chaining */
				inkey[0] = key[0] ^ (R >> 16);
				inkey[1] = key[1] ^ (R >> 8);
				inkey[2] = key[2] ^ R;
				inkey[3] = key[3] ^ (L >> 24);
				inkey[4] = key[4] ^ (L >> 16);
				inkey[5] = key[5] ^ (L >> 8);
				inkey[6] = key[6] ^ L;
			}
		}


		t = L;  L = R;  R = t;  // swap cancel

		/* Output */
		*inout++ = L;
		*inout++ = R;

		/* Subsequent blocks after the first use a truncated key schedule: */
		keyRound = 2;
	}

	return;
}
```

This page is left intentionally blank.