MISTY, KASUMI and Camellia Cipher Algorithm Development

*by Mitsuru Matsui and Toshio Tokita**

This article introduces three symmetric key block-cipher algorithms, MISTY, KASUMI, and Camellia, designed based on cipher evaluation techniques developed by Mitsubishi Electric Corp. MISTY and Camellia are designed for both high security and high speed/small size purposes. KASUMI is a cipher based on MISTY, and has recently been adopted as the standard cipher for mobile telephones in Europe.

MISTY and the Design Intent Behind it

MISTY is the family name for two 64-bit block-cipher algorithms, MISTY1 and MISTY2, that have 128-bit keys, designed by the corporation with detailed specifications announced in academic conferences in 1996 and 1997.[1][2]

In terms of security, MISTY has the major benefit of “provable security,” in which the security is proven mathematically against differential cryptanalysis and linear cryptanalysis, which are extremely powerful methods for breaking block ciphers. Through its use of the new round-function and recursive structures, MISTY is able to provide provable security while at the same time providing increased speed enabled by a heightened level of parallelism in internal components.

One major benefit of MISTY is that it is deployed in hardware. At the time, most ciphers were envisioned as being implemented in software, and as a result the hardware became extremely large, for otherwise, when implemented in software, there would be little hope for increases in speed. In contrast, the structure of the overall algorithm in MISTY uses logical functions and table-lookup procedures only, meaning that the resulting structure can be implemented aggressively using hardware characteristics, such as optimizing the structures of the tables for implementation in hardware.

The MISTY1 and MISTY2 specifications support a variable number of rounds (up to any

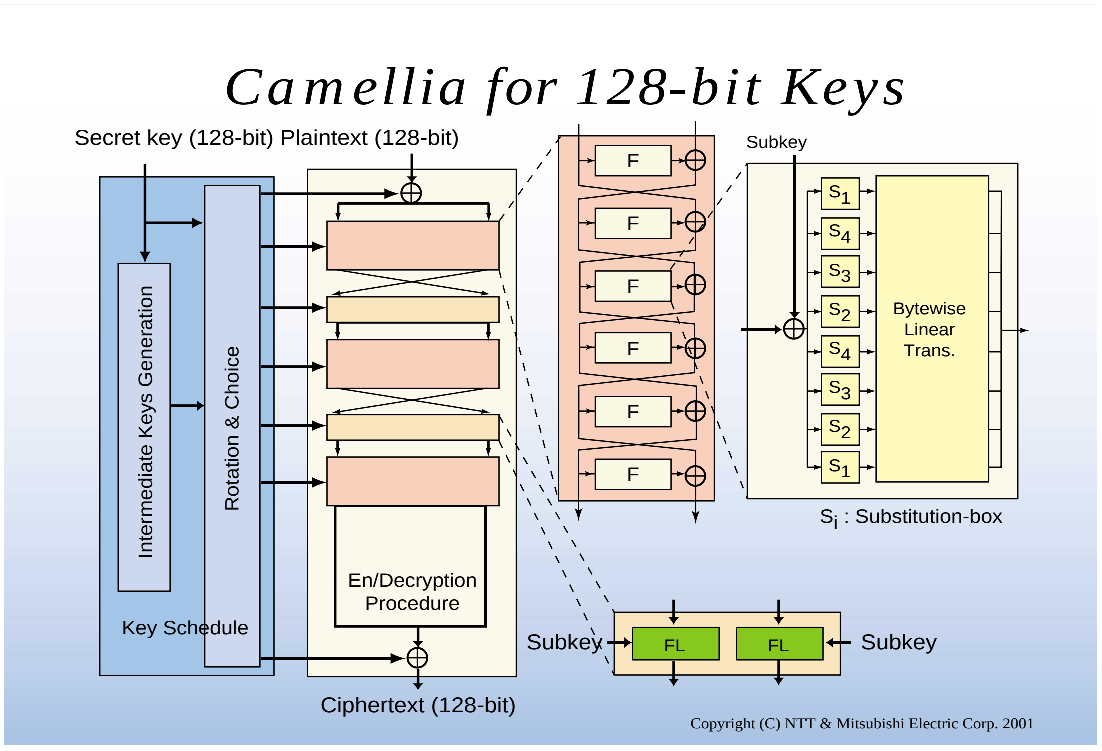

Fig. 1. The Camellia block-cipher algorithm

**Mitsuru Matsui and Toshio Tokita are with the Information Technology R&D Center* multiples of four); for MISTY1, eight rounds is recommended and for MISTY2, twelve rounds. At present, most implementations of MISTY1 use an eight-round version, and in the description below, any references to "MISTY" refers to the eight-round version of MISTY1, unless otherwise noted.

MISTY Today

MISTY has won a large number of users since its announcement. Along with the use of MISTY in a variety of software products for general use (such as an encryption library (PowerMISTY), file encryption (CryptoDoc), and email encryption (CryptoSign)), the corporation has also used MISTY in many governmental systems.

In terms of security, MISTY has been scrutinized by many researchers since its announcement. The reliability of an encryption method can only be established through the cumulative effect of third-party evaluations. According to a report by CRYPTREC, the Cryptography Research & Evaluation Committees for Japanese "e-government," at present there are no problems with the security of the 8-round MISTY1.[3]

KASUMI and its History

The 3GPP consortium, which discusses the technical standards for third-generation mobile telephone (W-CDMA) comprises the communication standards bodies from Europe, Japan, the United States, Korea, and China, and as part of this consortium, the SA-WG2 is a working group commissioned to establish standards for security architecture. Up to now, various communications ciphering methods used in Europe, including an encryption algorithm for second-generation mobile telephones, have been designed by the Security Algorithms Group of Experts (SAGE) as part of the European Telecommunications Standards Institute (ETSI), which is a 3GPP member, and the SA-WG2, reviewing these methods, requested SAGE to design the ciphering algorithm for the third-generation mobile telephone.

Having received this commission, SAGE set to work at designing the cipher, but because of factors such as the tight deadline for the development, SAGE decided to perform its development work based on an existing cipher, for which it selected MISTY1. The reason for adopting MISTY1 was not just its high level of security; the deciding factor was rather that MISTY1 was essentially the only block cipher that had been implemented at the time that fulfilled the required specification from the 3GPP side that it could be built in hardware using no more than 10K gates.

This 64-bit block cipher with a 128-bit key, designed based on MISTY1 was dubbed "KASUMI," the Japanese word for "misty," and was certified formally as the mandatory cipher in W-CDMA in January of 2000.

The Scope of W-CDMA Cipher Standards

The use of KASUMI as the mandatory standard is due to two mechanisms: the confidentiality and the integrity of its transport layer. When it comes to authentication, although each carrier may use a different method, the recommended (though not mandatory) method was, of course, designed by SAGE.

KASUMI is a variant of MISTY1 customized further for hardware, designed based on the assumption that it will be implemented in hardware (LSI circuits).

The KASUMI specifications have been publicly disclosed, and can be downloaded from the 3GPP home page.[4] Along with its widespread use in third-generation portable telephones in the future, KASUMI will find uses throughout the world. In July of 2002, GSM, the current generation of mobile telephone system in Europe, also made the decision to adopt KASUMI.

Camellia and the Design Intent Behind it.

Camellia is a new block-cipher algorithm developed jointly between NTT and Mitsubishi Electric in the year 2000. It was created by combining the world-class cipher strength evaluation technologies and cipher implementation technologies possessed by the two firms. The block size was set at 128 bits, twice that of MISTY and KASUMI.[5][6] The cipher supports three types of keys, 128-bit keys, 192-bit keys, and 256-bit keys.

Camellia, from the perspective of security, was not only designed to be resistant to the newest cryptanalysis methods including differential cryptanalysis, linear cryptanalysis, and beyond, but was also designed with a large safety margin in view of anticipated future progress in cryptanalysis techniques. Furthermore, in consideration of the broadening range of applications of ciphers in the recent past, Camellia has been designed so that it can be applied to all types of encryption platforms, enabling small/low-power applications in hardware for, for example, portable devices, and suitable for applications ranging from environments with extremely limited resources (such as IC cards) to the newest high-speed 64-bit processors.

Structural Features of Camellia

Similar to MISTY and KASUMI, Camellia is structured with only table lookup and logical operations, and thus it does not use any arithmetic operations, a structure adopted with the performance when implemented in hardware in mind. On the other hand, Camellia differs from MISTY and KASUMI in that the entire algorithm is structured with only byte- (or word-) unit operations, allowing high performance when implemented in software, regardless of the type of processor.

Camellia has been submitted to standardization in ISO, NESSIE, and the like, and the results of many third-party evaluations of its security are known. The CRYPTREC report also states that, at this time, there are no problems with the security of Camellia.[3] Camellia is expected to go into widespread use as a next-generation cipher. More information about the performance of Camellia is available on the Camellia web site at http://info.isl.ntt.co.jp/camellia.

This article has described the block-cipher algorithms, MISTY, KASUMI, and Camellia. Appended to the article is a CBC-mode sample program for MISTY1, written in the C language. This program was designed with portability in mind, and thus will work on most ANSI-C compilers. ❑

References

[1] Mitsuru Matsui: "Block Encryption MISTY" [Communications Science and Techniques] ISEC96-11 (1996).

[2] Mitsuru Matsui: "New Block Encryption Algorithm MISTY," FSE'97, Springer LNCS 1267 (1997).

[3] IPA (Information-technology Promotion Agency Security Center: Cipher Technology Evaluation Reports CRYPTREC Report 2000 (2001).

[4] 3GPP homepage http://www.3gpp.org/

[5] Aoki, et al: "The 128-Bit Block Cipher Camellia," [Communications Science and Techniques] ISEC2000-6 (2000).

[6] Aoki, et al: "The 128-Bit Block Cipher Camellia," IEICE Trans. Fundamentals, vol. E85-A no.1 (2001).

```
/*********************************************************************
 A Sample Program of MISTY1 Block Encryption Algorithm in CBC mode

 Key Scheduling    P_Misty1_Keysch( key, ekey )
 Encryption   P_Misty1_Cbcenc( pdat, cdat, ivec, ekey, block )
 Decryption   P_Misty1_Cbcdec( cdat, pdat, ivec, ekey, block )

     key  address of encryption key (16 bytes)
     ekey address of subkey (sizeof(Uint) * 32 bytes)
     pdat address of plaintext data (block * 8 bytes)
     cdat address of ciphertext data (block * 8 bytes)
     ivec address of initial vector (8 bytes)
     block    the number of data blocks

     Copyright (c) 2002 Mitsubishi Electric Corporation
*********************************************************************/

typedef unsigned char UInt8;
typedef unsigned int UInt; /* also works for short and long */

static const UInt S7[128] = {
 27, 50, 51, 90, 59, 16, 23, 84, 91, 26,114,115,107, 44,102, 73,
 31, 36, 19,108, 55, 46, 63, 74, 93, 15, 64, 86, 37, 81, 28, 4,
 11, 70, 32, 13,123, 53, 68, 66, 43, 30, 65, 20, 75,121, 21,111,
 14, 85, 9, 54,116, 12,103, 83, 40, 10,126, 56, 2, 7, 96, 41,
 25, 18,101, 47, 48, 57, 8,104, 95,120, 42, 76,100, 69,117, 61,
 89, 72, 3, 87,124, 79, 98, 60, 29, 33, 94, 39,106,112, 77, 58,
 1,109,110, 99, 24,119, 35, 5, 38,118, 0, 49, 45,122,127, 97,
 80, 34, 17, 6, 71, 22, 82, 78,113, 62,105, 67, 52, 92, 88,125 };

static const UInt S9[512] = {
451,203,339,415,483,233,251, 53,385,185,279,491,307, 9, 45,211,
199,330, 55,126,235,356,403,472,163,286, 85, 44, 29,418,355,280,
331,338,466, 15, 43, 48,314,229,273,312,398, 99,227,200,500, 27,
1,157,248,416,365,499, 28,326,125,209,130,490,387,301,244,414,
467,221,482,296,480,236, 89,145, 17,303, 38,220,176,396,271,503,
231,364,182,249,216,337,257,332,259,184,340,299,430, 23,113, 12,
71, 88,127,420,308,297,132,349,413,434,419, 72,124, 81,458, 35,
317,423,357, 59, 66,218,402,206,193,107,159,497,300,388,250,406,
481,361,381, 49,384,266,148,474,390,318,284, 96,373,463,103,281,
101,104,153,336, 8, 7,380,183, 36, 25,222,295,219,228,425, 82,
265,144,412,449, 40,435,309,362,374,223,485,392,197,366,478,433,
195,479, 54,238,494,240,147, 73,154,438,105,129,293, 11, 94,180,
329,455,372, 62,315,439,142,454,174, 16,149,495, 78,242,509,133,
253,246,160,367,131,138,342,155,316,263,359,152,464,489, 3,510,
189,290,137,210,399, 18, 51,106,322,237,368,283,226,335,344,305,
327, 93,275,461,121,353,421,377,158,436,204, 34,306, 26,232, 4,
391,493,407, 57,447,471, 39,395,198,156,208,334,108, 52,498,110,
202, 37,186,401,254, 19,262, 47,429,370,475,192,267,470,245,492,
269,118,276,427,117,268,484,345, 84,287, 75,196,446,247, 41,164,
14,496,119, 77,378,134,139,179,369,191,270,260,151,347,352,360,
215,187,102,462,252,146,453,111, 22, 74,161,313,175,241,400, 10,
426,323,379, 86,397,358,212,507,333,404,410,135,504,291,167,440,
321, 60,505,320, 42,341,282,417,408,213,294,431, 97,302,343,476,
114,394,170,150,277,239, 69,123,141,325, 83, 95,376,178, 46, 32,
469, 63,457,487,428, 68, 56, 20,177,363,171,181, 90,386,456,468,
24,375,100,207,109,256,409,304,346, 5,288,443,445,224, 79,214,
319,452,298, 21, 6,255,411,166, 67,136, 80,351,488,289,115,382,
188,194,201,371,393,501,116,460,486,424,405, 31, 65, 13,442, 50,
61,465,128,168, 87,441,354,328,217,261, 98,122, 33,511,274,264,
448,169,285,432,422,205,243, 92,258, 91,473,324,502,173,165, 58,
459,310,383, 70,225, 30,477,230,311,506,389,140,143, 64,437,190,
120, 0,172,272,350,292, 2,444,162,234,112,508,278,348, 76,450 };
```

```
#define FL_enc( k ){\
r1 ^= r0 & ekey[0+((k+0)&7)];\
r3 ^= r2 & ekey[8+((k+2)&7)];\
r0 ^= r1 | ekey[8+((k+6)&7)];\
r2 ^= r3 | ekey[0+((k+4)&7)];\
}

#define FL_dec( k ){\
r0 ^= r1 | ekey[0+((k+4)&7)];\
r2 ^= r3 | ekey[8+((k+6)&7)];\
r1 ^= r0 & ekey[8+((k+2)&7)];\
r3 ^= r2 & ekey[0+((k+0)&7)];\
}

#define FI_key( k ){\
r0 = ekey[k] >> 7;\
r1 = ekey[k] & 0x7f;\
r0 = S9[r0] ^ r1;\
r1 = S7[r1] ^ ( r0 & 0x7f );\
r1 ^= ekey[(k+1)&7] >> 9;\
r0 ^= ekey[(k+1)&7] & 0x1ff;\
r0 = S9[r0] ^ r1;\
ekey[ 8+k] = r1 << 9 ^ r0;\
ekey[16+k] = r0;\
ekey[24+k] = r1;\
}

#define FI_txt( a0, a1, k ){\
a1 = a0 >> 7;\
a0 &= 0x7f;\
a1 = S9[a1] ^ a0;\
a0 = S7[a0] ^ a1;\
a1 ^= ekey[16+(k)];\
a0 ^= ekey[24+(k)];\
a0 &= 0x7f;\
a1 = S9[a1] ^ a0;\
a1 ^= a0 << 9;\
}

#define FO_txt( a0, a1, a2, a3, k ){\
t0 = a0 ^ ekey[k];\
FI_txt( t0, t1, (k+5)&7 );\
t1 ^= a1;\
t2 = a1 ^ ekey[(k+2)&7];\
FI_txt( t2, t0, (k+1)&7 );\
t0 ^= t1;\
t1 ^= ekey[(k+7)&7];\
FI_txt( t1, t2, (k+3)&7 );\
t2 ^= t0;\
t0 ^= ekey[(k+4)&7];\
a2 ^= t0;\
a3 ^= t2;\
}

void P_Misty1_Keysch( UInt8 *key, UInt *ekey )
{
UInt r0,r1;

ekey[0] = (UInt)key[ 0]<<8 ^ (UInt)key[ 1];
ekey[1] = (UInt)key[ 2]<<8 ^ (UInt)key[ 3];
ekey[2] = (UInt)key[ 4]<<8 ^ (UInt)key[ 5];
ekey[3] = (UInt)key[ 6]<<8 ^ (UInt)key[ 7];
```

```
ekey[4] = (UInt)key[ 8]<<8 ^ (UInt)key[ 9];
ekey[5] = (UInt)key[10]<<8 ^ (UInt)key[11];
ekey[6] = (UInt)key[12]<<8 ^ (UInt)key[13];
ekey[7] = (UInt)key[14]<<8 ^ (UInt)key[15];

FI_key( 0 ); FI_key( 1 ); FI_key( 2 ); FI_key( 3 );
FI_key( 4 ); FI_key( 5 ); FI_key( 6 ); FI_key( 7 );
}

void P_Misty1_Cbcenc( UInt8 *pdat, UInt8 *cdat, UInt8 *ivec, UInt *ekey, UInt
block )
{
UInt r0,r1,r2,r3,t0,t1,t2,buff[4];

buff[0] = (UInt)ivec[0]<<8 ^ (UInt)ivec[1];
buff[1] = (UInt)ivec[2]<<8 ^ (UInt)ivec[3];
buff[2] = (UInt)ivec[4]<<8 ^ (UInt)ivec[5];
buff[3] = (UInt)ivec[6]<<8 ^ (UInt)ivec[7];

while( block != 0 ){
  r0 = (UInt)pdat[0]<<8 ^ (UInt)pdat[1];
  r1 = (UInt)pdat[2]<<8 ^ (UInt)pdat[3];
  r2 = (UInt)pdat[4]<<8 ^ (UInt)pdat[5];
  r3 = (UInt)pdat[6]<<8 ^ (UInt)pdat[7];

  r0 ^= buff[0]; r1 ^= buff[1];
  r2 ^= buff[2]; r3 ^= buff[3];

  FL_enc( 0 );
  FO_txt( r0, r1, r2, r3, 0 );
  FO_txt( r2, r3, r0, r1, 1 );
  FL_enc( 1 );
  FO_txt( r0, r1, r2, r3, 2 );
  FO_txt( r2, r3, r0, r1, 3 );
  FL_enc( 2 );
  FO_txt( r0, r1, r2, r3, 4 );
  FO_txt( r2, r3, r0, r1, 5 );
  FL_enc( 3 );
  FO_txt( r0, r1, r2, r3, 6 );
  FO_txt( r2, r3, r0, r1, 7 );
  FL_enc( 4 );

  buff[0] = r2; buff[1] = r3;
  buff[2] = r0; buff[3] = r1;

  cdat[0] = (UInt8)(r2 >> 8); cdat[1] = (UInt8)(r2);
  cdat[2] = (UInt8)(r3 >> 8); cdat[3] = (UInt8)(r3);
  cdat[4] = (UInt8)(r0 >> 8); cdat[5] = (UInt8)(r0);
  cdat[6] = (UInt8)(r1 >> 8); cdat[7] = (UInt8)(r1);

  pdat += 8; cdat += 8; block--;
}

ivec[0] = (UInt8)(buff[0] >> 8); ivec[1] = (UInt8)(buff[0]);
ivec[2] = (UInt8)(buff[1] >> 8); ivec[3] = (UInt8)(buff[1]);
ivec[4] = (UInt8)(buff[2] >> 8); ivec[5] = (UInt8)(buff[2]);
ivec[6] = (UInt8)(buff[3] >> 8); ivec[7] = (UInt8)(buff[3]);
}

void P_Misty1_Cbcdec( UInt8 *cdat, UInt8 *pdat, UInt8 *ivec, UInt *ekey, UInt
block )
{
```

```
UInt r0,r1,r2,r3,t0,t1,t2,buff[4],buff2[4];

buff[0] = (UInt)ivec[0]<<8 ^ (UInt)ivec[1];
buff[1] = (UInt)ivec[2]<<8 ^ (UInt)ivec[3];
buff[2] = (UInt)ivec[4]<<8 ^ (UInt)ivec[5];
buff[3] = (UInt)ivec[6]<<8 ^ (UInt)ivec[7];

while( block != 0 ){
  r0 = (UInt)cdat[0]<<8 ^ (UInt)cdat[1];
  r1 = (UInt)cdat[2]<<8 ^ (UInt)cdat[3];
  r2 = (UInt)cdat[4]<<8 ^ (UInt)cdat[5];
  r3 = (UInt)cdat[6]<<8 ^ (UInt)cdat[7];

  buff2[0] = r0; buff2[1] = r1;
  buff2[2] = r2; buff2[3] = r3;

  FL_dec( 4 );
  FO_txt( r0, r1, r2, r3, 7 );
  FO_txt( r2, r3, r0, r1, 6 );
  FL_dec( 3 );
  FO_txt( r0, r1, r2, r3, 5 );
  FO_txt( r2, r3, r0, r1, 4 );
  FL_dec( 2 );
  FO_txt( r0, r1, r2, r3, 3 );
  FO_txt( r2, r3, r0, r1, 2 );
  FL_dec( 1 );
  FO_txt( r0, r1, r2, r3, 1 );
  FO_txt( r2, r3, r0, r1, 0 );
  FL_dec( 0 );

  r2 ^= buff[0]; r3 ^= buff[1];
  r0 ^= buff[2]; r1 ^= buff[3];

  buff[0] = buff2[0]; buff[1] = buff2[1];
  buff[2] = buff2[2]; buff[3] = buff2[3];

  pdat[0] = (UInt8)(r2 >> 8); pdat[1] = (UInt8)(r2);
  pdat[2] = (UInt8)(r3 >> 8); pdat[3] = (UInt8)(r3);
  pdat[4] = (UInt8)(r0 >> 8); pdat[5] = (UInt8)(r0);
  pdat[6] = (UInt8)(r1 >> 8); pdat[7] = (UInt8)(r1);

  pdat += 8; cdat += 8; block--;
}

ivec[0] = (UInt8)(buff[0] >> 8); ivec[1] = (UInt8)(buff[0]);
ivec[2] = (UInt8)(buff[1] >> 8); ivec[3] = (UInt8)(buff[1]);
ivec[4] = (UInt8)(buff[2] >> 8); ivec[5] = (UInt8)(buff[2]);
ivec[6] = (UInt8)(buff[3] >> 8); ivec[7] = (UInt8)(buff[3]);
}
```