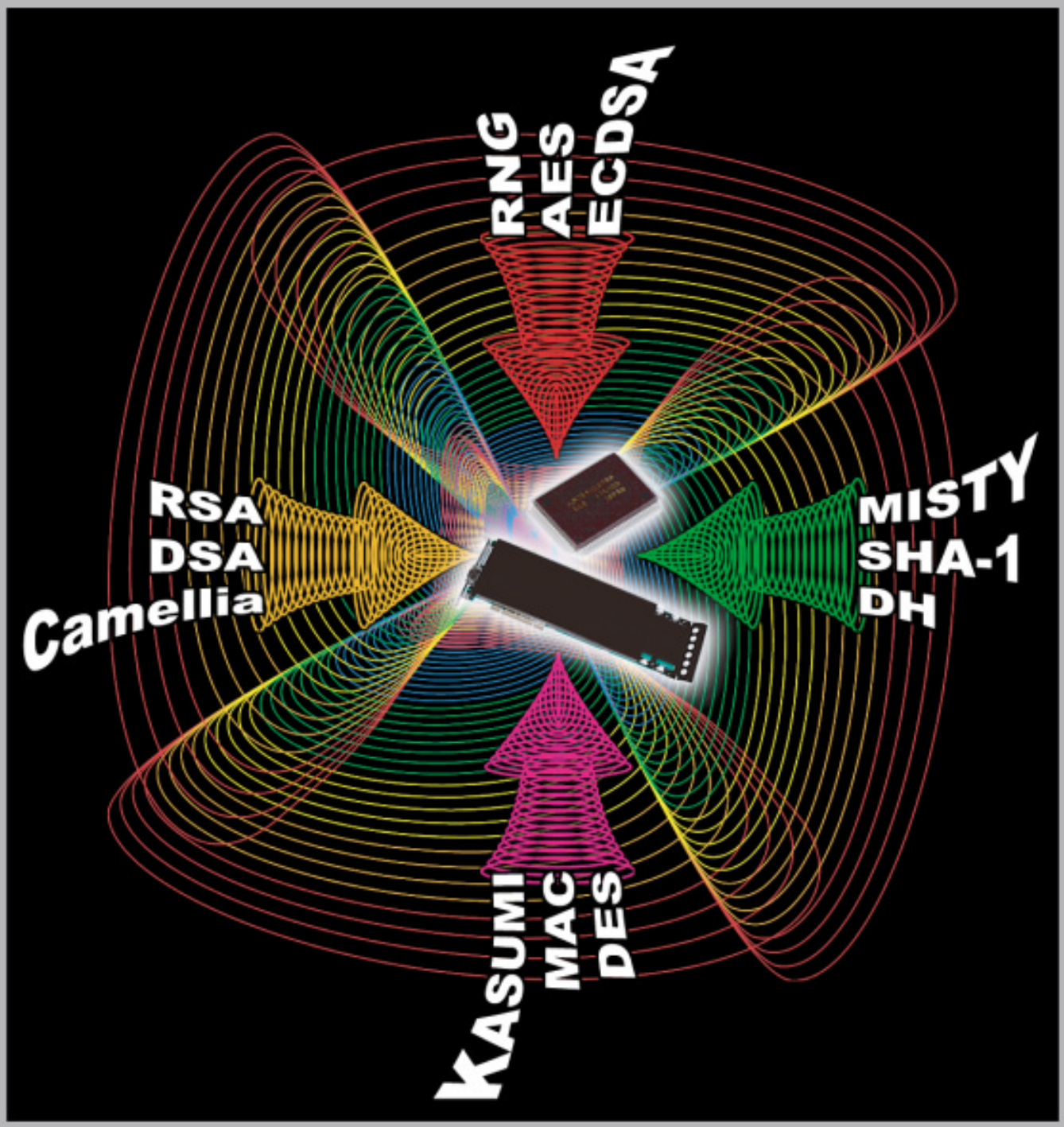
ISSN 1345-3041

VOL. 100/DEC. 2002

MITSUBISHI ELECTRIC ADVANCE

Cryptography Edition



MITSUBISHI ELECTRIC

Changes for the Better

A Quarterly Survey of New Products, Systems, and Technology

●Vol. 100/December 2002 Mitsubishi Electric **ADVANCE**

Cryptography Edition

CONTENTS

TECHNICAL REPORTS

Cover Story
Our cryptographic technology is not limited to our proprietary encryption algorithm development but also provides widespread support to industry standardization activity. There are two kinds of information security products in the front cover implementing our symmetric key cryptographic algorithm MISTY. The upper one is a typical cryptographic LSI and the lower one is the tamper-resistant secure board. Details of these and other products are provided in the articles of this special edition of *Advance*.

Editorial Inquiries

Keizo Hama
Corporate Total Productivity Management & Environmental Programs
Mitsubishi Electric Corporation
2-2-3 Marunouchi
Chiyoda-ku, Tokyo 100-8310, Japan
Fax 03-3218-2465

Product Inquiries

Planning & Coordination Dept.
Information Technology R&D Center
Mitsubishi Electric Corporation
misty@isl.melco.co.jp
http://www.mitsubishielectric.co.jp/security/

Mitsubishi Electric Advance is published on line quarterly (in March, June, September, and December) by Mitsubishi Electric Corporation.

Overview

The Current State of Encryption Technology

*by Kotaro Katsuyama**

With advances in the Internet, progress in mobile phones and other mobile communications devices, and with similar developments taking place in digital electrical equipment for the home and in sensors, we are entering the era of the ubiquitous network. While this offers society the prospect of great convenience, enabling the almost instantaneous exchange of large volumes of electronic data, the facts that everything will be connected to the same network and communications will be possible from any one point to any other within the network pose new threats of eavesdropping, tampering, impersonation and breaches of security. These threats are making the technology for ensuring the security of information, particularly encryption technology, increasingly important. Mitsubishi Electric possesses world-class technologies in this area: our MISTY encryption technology is used in the KASUMI algorithm adopted for third-generation mobile phone systems and for GSM. The corporation also provides a number of advanced security solutions that make major contributions to the safety and security of the social infrastructure.

This issue of *Advance* introduces the corporation's MISTY, Camellia and KASUMI encryption algorithms, along with the technologies for evaluating the degree of security provided, and for implementing these algorithms. The tamper-proof TURBOMISTY secure boards are also introduced. Finally, there is an article that outlines the corporation's work directed at future applications of quantum cryptography. ❑

**Kotaro Katsuyama is with the Information Technology R&D Center*

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

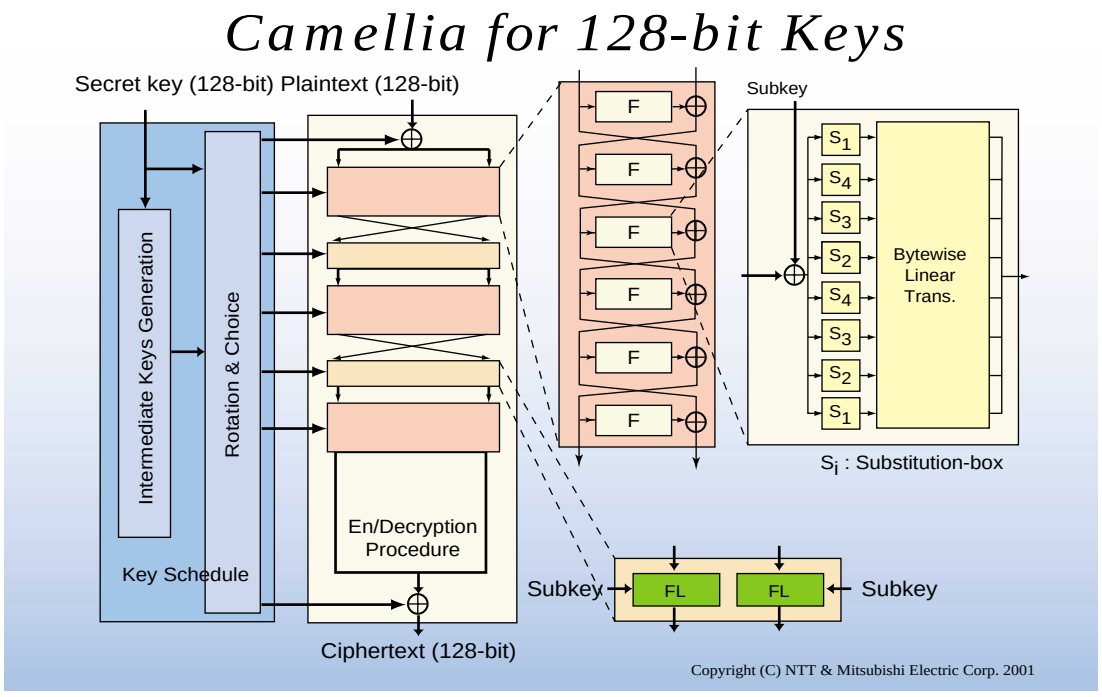


Fig. 1. The Camellia block-cipher algorithm

**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]);
}
```

Cryptanalysis Technique to Evaluate the Strength of Ciphers

*by Toshio Tokita, Yasuyuki Sakai and Katsuyuki Takashima**

The design of secure ciphers (cryptosystems) generally requires cryptoanalytical techniques with which to evaluate their strength. This paper provides a summary of methods of evaluating the strengths of common-key block ciphers and public-key cryptosystems. It also introduces software developed by Mitsubishi Electric Corporation to evaluate the security and the performance of cryptosystems.

Common-Key Block-Cipher Strength Evaluation

Here we explain differential and linear cryptanalysis, methods commonly used for common-key block ciphers.

DIFFERENTIAL CRYPTANALYSIS AND DIFFERENTIAL CHARACTERISTIC PROBABILITIES. Differential cryptanalysis, proposed by Biham, et al, in 1990 is based on the principle that it is possible to characterize statistically variations between plaintexts and ciphertexts, where the characteristics can be used to guess key information. Generally, the cipher strength against this cryptanalysis method is expressed in terms of maximum average differential probability, as in Eq. 1, with lower maximum average differential probability indicating more secure ciphers. Here, ΔP ($\neq 0$) and ΔC are the amounts of change in plaintext P and ciphertext C, respectively, where (+) indicates a bitwise exclusive OR.

$$DP_{max} = \max_{\Delta P \neq 0,\ \Delta C} \text{Prob}\{F(P + \Delta P) + F(P) = \Delta C\} \quad \text{..... Eq. 1}$$

However, for any given encryption algorithm the accurate calculation of the DP_{max} value is extremely difficult because of the computational complexity involved. Given this, instead of performing the calculation, the algorithm $C = F(P)$ is broken down into small component functions F_1, F_2, F_3,..., in the form of $C = F_n(...(F_2(F_1(P))))$, where generally the maximum differential characteristic probability as defined in Eq. 2 is used as the indicator for the strength against the differential cryptanalysis method.

$$DP'_{max} = \max \Pi\ \text{Prob}\{F_i(P_i + \Delta P_i) + F_i(P_i) = \Delta P_{i+1}\} \quad \text{..... Eq. 2}$$

On the other hand, the number of plaintext and ciphertext pairs required for success with differential cryptanalysis is inversely proportional to the maximum differential characteristic probability, where the smaller this probability value, the more secure the encryption.

LINEAR CRYPTANALYSIS AND LINEAR CHARACTERISTIC PROBABILITIES. Linear cryptanalysis was proposed by the corporation in 1993, based on the principle that it is possible to characterize statistically the relationship between plaintexts, ciphertexts, and the bits in the key, where the characteristics can be used to guess key information. The strength of a cipher against linear cryptanalysis is expressed in terms of the maximum average linear probability as defined in Eq. 3, where the smaller the maximum average linear probability, the more secure the encryption.

Here ΓP and ΓC ($\neq 0$) indicate the mask values of the plaintext P and the ciphertext C, respectively, and $(\cdot)$ indicates the parity of the value that is calculated as the logical AND for each bit.

$$LP_{max} = \max_{\Gamma C \neq 0, \Gamma P} |2 \cdot \text{Prob}\{P \cdot \Gamma P = C \cdot \Gamma C\} - 1|^2 \quad \text{..... Eq. 3}$$

However, for any given encryption algorithm the accurate calculation of the LP_{max} value is extremely difficult because of the computational complexity involved. Given this, instead of performing the calculation, the algorithm $C = F(P)$ is broken down into small component functions F_1, F_2, F_3,..., in the form of $C = F_n(...(F_2(F_1(P))))$, where generally the maximum differential characteristic probability as defined in Eq. 4 is used as the indicator for the strength of the linear cryptanalysis.

$$LP'_{max} = \max \Pi\ |2 \cdot \text{Prob}\{P_i \cdot \Gamma P_i = P_{i+1} \cdot \Gamma P_{i+1}\} - 1|^2 \quad \text{..... Eq. 4}$$

On the other hand, the number of plaintext and ciphertext pairs required for success in linear cryptanalysis is inversely proportional to the maximum differential characteristic probability,

**Toshio Tokita, Yasuyuki Sakai and Katsuyuki Takashima are with the Information Technology R&D Center.*

where the smaller this probability value, the more secure the encryption.

OTHER STRENGTH EVALUATIONS. See reference [2] for other cryptanalytical methods not described here (for example, truncated differential cryptanalysis, higher-order cryptanalysis, etc).

There is also the case where a common-key block cipher is used in a mode such as OFB for random-number generation. Generally, an evaluation of randomness in such cases requires statistical methods, considering long-period characteristics, linear complexity, equal 0/1 frequency, etc.

Public-Key Cryptosystem Strength Evaluations

Generally, public-key cryptosystems are designed basing their security on the intractability of the following problems in number theory:

1. The integer-factorization problem
2. The finite-field discrete-logarithm problem
3. The elliptic-curve discrete-logarithm problem

Here, we call these "integer-factorization based public-key cryposystems," "discrete-logarithm based public-key cryptosystems" and "elliptic curve discrete-logarithm based public-key cryptosystems" and explain each of them below.

INTEGER-FACTORIZATION BASED PUBLIC-KEY CRYPTOSYSTEM STRENGH EVALUATIONS. The RSA cryptosystem, which is the most common public-key scheme in use today, is founded on the security provided by the intractability of the integer-factorization problem. The most obvious way to attack the RSA cryptosystem is by factorizing the publicly known composite modulus n (the product of two distinct primes that are themselves private information). There is a considerable literature on factoring algorithms.

The running time of some factoring algorithms depends solely on the size of n. They include the quadratic-sieve method and the number-field sieve method.

In contrast, some algorithms are tailored to perform better when the composite modulus n is of a special type. The running times of such algorithms therefore typically depend on certain properties of the factors of n. They include Pollard's rho method , the elliptic curve method, the p-1 method, and the p+1 method.

Although the difficulty of solving the integer-factorization problem typically increases with the size of the composite number (with the computational complexity increasing in an order termed sub-exponential time), when the composite is the product of prime factors that have certain properties, then the factorization can be done quickly even if the modulus itself is large.

DISCRETE-LOGARITHM BASED PUBLIC-KEY CRYPTOSYSTEM STRENGTH EVALUATIONS. The discrete-logarithm problem is one that is used broadly in parallel with the factorization problem. For a given prime number p, a generator g of the multiplicative group of Z_p, and an element y of the multiplicative group of Z_p, the problem is to find an integer x such that $y \equiv g^x \bmod p$.

Here, x is known as the base-p discrete logarithm. For example, x is 4 when $4 \equiv 3^x \bmod 7$. Although logarithm calculations for real numbers are easy, the logarithm calculations for base p, or in other words, logarithmic calculations in a discrete domain are difficult when p is large.

The ElGamal cryptosystem is typical of those that base their security on the intractability of this problem. Additionally, even in the most common key-agreement protocol, the Diffie-Hellman (DH) protocol, advantage is taken of the fact that the discrete-logarithm problem is difficult to solve. (Note that in the DH key-agreement protocol, security is based on the DH assumption, which is a stronger assumption than the discrete-logarithm assumption.)

One possible attack on the ElGamal cryptosystem and the DH key-agreement protocol is to calculate the private information x from the public information y, g, and p. Such methods include Pollar's rho method, the Pohlig-Hellman method, the index-calculus method, and the number-field sieve method. The discrete-logarithm problem, like the factorization problem, is generally more difficult to solve when the various parameters are larger (that is, the number of calculations required increases in what is termed sub-exponential time); however, when parameters having specific characteristics are selected, the discrete-logarithm calculations can be done easily regardless of the size of the parameters.

ELLIPTIC-CURVE DISCRETE-LOGARITHM PUBLIC-KEY CRYPTOSYSTEM STRENGTH EVALUATIONS. Algorithms for RSA and ElGamal cryptosystems use integer (or more precisely, "finite field") additions and multiplications. Similarly, the elliptic-curve cryptosystem is of the same type in that it uses addition on an elliptic curve. (In Fig. 1, point S is the result of adding point Q and point R.) The computational complexity problem of the elliptic curve discrete logarithm is formulated in terms of addition on an elliptic curve, and the elliptic-curve crptosystem, as typified by the

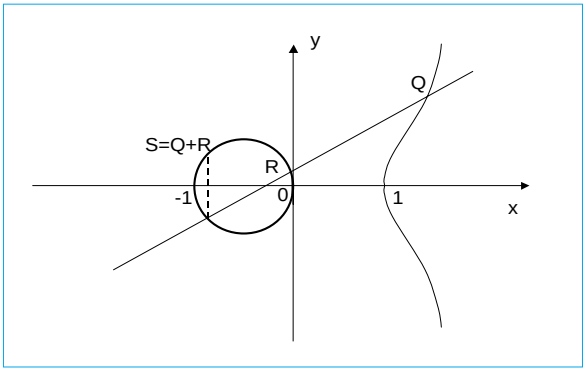


Fig. 1 Example of elliptic curve : $y^2 = x^3 - x$

elliptic-curve ElGamal cryptosystem, bases its security on that difficulty. Generally no algorithms have been established by which to solve the elliptic-curve discrete-logarithm problem efficiently, but there are efficient cryptanalytic methods for specific elliptic curves.

Methods for attacking specific elliptic-curve cryptosystem parameters include the Pohlig-Hellman method, the MOV method, the FR method, and the Satoh-Araki-Semaev-Smart (SASS) method. In order to preserve sufficient security against the MOV method or the FR method, the MOV (FR) reduction degree of the elliptic curve must be large. Additionally, in order to be secure against the SASS method, the trace of the Frobenius endomorphism on the elliptic curve must not be one. Furthermore, in order to be secure from the Pohlig-Hellman method, the number of rational points on the elliptic curve must be a (pseudo-) prime. Establishing the parameters of the elliptic-curve cryptosystem to fulfill these criteria requires the number of rational points on the elliptic curve to be calculated with some ingenuity. The so-called SEA method, which places no restrictions on the characteristics of finite fields, has been well established for such calculations for over eight years. On the other hand, the recently-established Satoh method is particularly useful when the characteristic is small, and the Skjernaa, FGH and AGM methods are improvements of it.

These fast algorithms are based on number theory, and the search for improved algorithms continues. Recently the Weil-descent method (and in particular, the GHS method) has been discovered as one method of cryptographic attack, and as a result, the current recommendation for protection is to use a group of rational points on an elliptic curve, whose values of (x, y) coordinates are in a prime degree extension field over prime field.

Cipher-Performance Analysis Software

This section will discuss the cipher-performance analysis software developed by Mitsubishi Electric. This is divided into software for common-key and public-key cryptosystems, and evaluates the strength of a cipher against any of the methods discussed above.

COMMON-KEY CRYPTOSYSTEM PERFORMANCE EVALUATION SOFTWARE. Fig. 2 shows the structure of an approach to common-key cryptosystem performance analysis software. Summaries will be given for each of the software components comprising the common-key cryptosystem performance evaluation software.

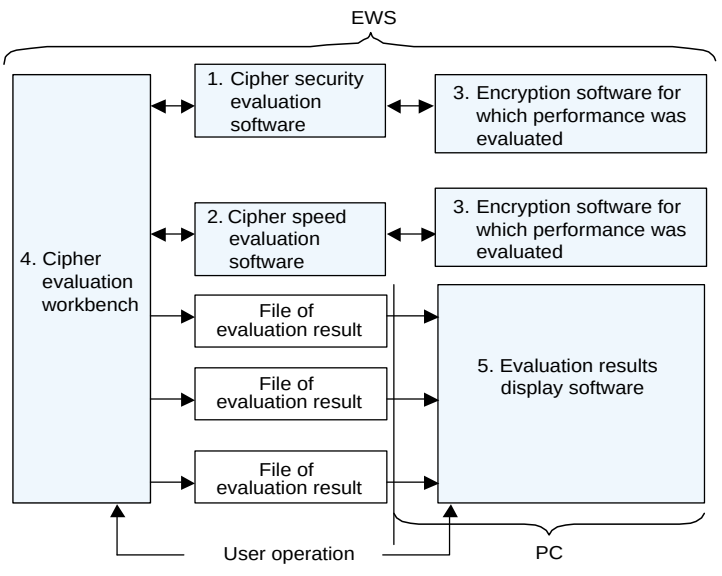


Fig. 2 Evaluation software for common-key cryptosystem (outline)

1. Cipher security-evaluation software
   This software evaluates the differential characteristic probabilities and the linear characteristic probabilities of the cipher to be evaluated. Additionally, when a pseudo-random number generator is used, it investigates frequency tests as a measure of randomness, and investigates collision tests and linear complexity.

2. Cipher speed-evaluation software
   This evaluates the processing speed (as an absolute value) of the encryption/decryption processes on a specific platform for the algorithm to be evaluated. Virtual platforms can also be designated and the processing speeds will be evaluated as a relative speed, relative to the virtual platform

3. Encryption software for which performance was evaluated
   The encryption algorithms for the ciphers to be evaluated had functions for absolute speed evaluations and security evaluations for AES (Rijndael), Serpent, CAST-256, and Twofish, and functions that were subjected to relative speed evaluations.

4. Cipher-evaluation workbench
   GUI functions are used when setting the evaluation parameters.

5. Evaluation-results display software
   GUI functions are used when displaying the evaluation results.

PUBLIC-KEY CRYPTOSYSTEM PERFORMANCE EVALUATION SOFTWARE. Fig. 3 shows the structure of an approach to public-key cryptosystem performance analysis software. A brief summary of the various software components comprising the public-key cryptosystem security evaluation software is presented below.

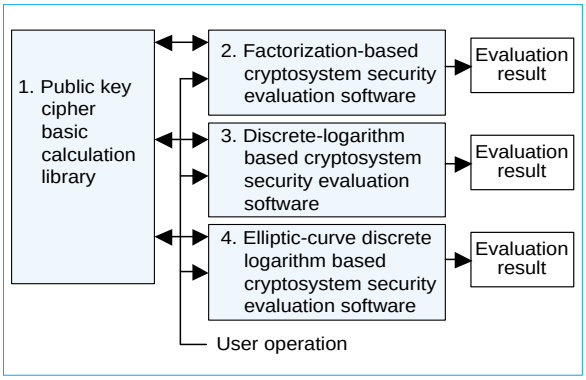


Fig. 3 Evaluation software for public-key cryptosystem (outline)

1. Public-key cryptosystem basic calculation library
   This is a library of basic calculations for public-key cryptosystems.

2. Security evaluation software for integer-factorization based cryptosystems
   This performs security analysis for integer-factorization based public-key cryptosystems using the quadratic-sieve and elliptic-curve methods of factoring.

3. Security-evaluation software for discrete-logarithm based cryptosystems
   This performs security analysis for discrete-logarithm based public-key cryptosystems using the Pohlig-Hellman and index-calculus methods.

4. Security-evaluation software for elliptic-curve discrete-logarithm based cryptosystems
   This performs security analysis for elliptic-curve discrete-logarithm based public-key cryptosystems based on counting the number of rational points on an elliptic curve (SEA method), on the trace of the Frobenius endomorphism on the elliptic curve, and on the MOV conditions.

Cipher-evaluation technologies are progressing daily based on innovations in cryptanalysis methods. As a result, the cipher-strength evaluation software described in this paper will need to undergo enhancements to handle each new innovation. ❑

Note: The cipher performance evaluation software described in this paper includes results from the Cipher Strength Evaluation Technology Development Project by the Department for the Development of Fundamental Technologies for the Next-Generation Digital Industry, of the Information-Technology Promotion Agency, Japan.

References:

[1] Information-Technology Promotion Agency, Japan Security Center: Cipher Technology Evaluation Report: Cryptrec Report 2000 (2001-3)

[2] Communications Broadcast Mechanisms: Common key Block Cipher Selection/Design/Evaluation Documents (2000-6)

[3] Cohen, H. : A Course in Computational Algebraic Number Theory, Springer-Verlag (1993)

[4] Shin'ichi Amada, et al.: Cipher Performance Evaluation Software Development, SCIS2000-A51 (2000-1)

Cipher Algorithm Implementation

*by Junko Nakajima, Tetsuya Ichikawa and Tomomi Kasuya**

This article describes cipher-implementation technology. When implementing ciphers one must carefully consider the usage environment and security policies in order to strike a balance between implementation scope and performance. Two types of cipher implementation (in software and in hardware) are discussed here, and we conclude by touching upon reusable design assets (intellectual properties) in hardware implementation, which has been the focus of so much attention recently.

Implementation in Software

THE MISTY1 IMPLEMENTATION METHOD. Many techniques exist to increase the speed of ciphers in software. While we have included several of these techniques in MISTY1[1], we have also given thought to optimizing the programming based on the characteristics of specific processors.[2]~[5]

As variations on the method of implementing the MISTY1 algorithm, three of the following four methods (numbered below) use tables of different sizes. As shown for the fourth, the well-known "Bit Slice" implementation can also be applied to MISTY1 as a method well-suited to parallel processing. The numbers in parentheses indicate the sizes of the tables.

1. Implementation as shown in the specification drawings (2.3kbyte)
2. Reduced number of instructions using FI-function equivalent conversion (2.3kbyte)
3. Reduced number of instructions using dynamic table generation (9.5kbyte)
4. Multiple block parallel implementation using bit slice (0kbyte, i.e., zero table)

IMPLEMENTATION ON RISC PROCESSORS. In the second and third methods listed above, it is possible to reduce the total number of instructions without dramatically increasing the table size as a result of internal function equivalent conversion. When these methods are used on the Pentium*-II and -III, Alpha-21264, or similar processors, the table size will be less than the size of the primary cache, meaning that these methods provide a substantial increase in the processing speed in these processors.[6]&[7] The highest speed performance in a MISTY1 software implementation has been achieved using the parallel processing method in four above.

The key-schedule implementation also varies across these four methods. In particular, where tables depend on the key, the tables are generated by the key-schedule part, and thus the overhead of that part is large, and in actual use the time that is required to do so cannot be ignored. As a result, the most appropriate method should be selected according to the data size and the application after taking into account the effects of key scheduling time as well, see Tables 1 and 2.

Table 1 Alpha 21264 (667Mhz)

Deployment method	Table size	Expanded key generation (cycles/key)	Cipher speed (cycles/key)
[1]	2.6 kB	109	245
[2]	4.6 kB	200	197
[3]	18.9 kB	12450	192
[4]	--	17	68

Table 2 Pentium III (800Mhz)

Deployment method	Table size	Expanded key generation (cycles/key)	Cipher speed (cycles/key)
[1]	2.3 kB	124	219
[2]	2.3 kB	230	207
[3]	9.5 kB	8745	193
[4]	--	46	169

COMPARISONS WITH OTHER CIPHERS. Fair comparisons of software performance require not only the use of the same platform but also the same coding style and the same speed measurement program, and the fact is that, at present, the speeds of the various algorithms are measured on a variety of different platforms using a variety of different measurement methods. Completely standardizing these platforms and methods for the purposes of comparison is probably a practical impossibility. With this caveat,

**Junko Nakajima and Tomomi Kasuya are with the Information Technology R&D Center and Tetsuya Ichikawa is with Mitsubishi Electric Engineering Co., Ltd.*

Fig.1 shows the results of known high-speed encryption implementation, in terms of the number of encryption cycles per byte unit, not broken out by the platform used.

Here, the white bars show the speeds in normal implementation using single-block units, while the black bars show the speeds when parallel processing is allowed. In MISTY1, the speed for 64-bit ciphering in normal single-block implementation is about the same as that of other methods, but if parallel processing is not allowed, the speed is decisively faster in processors with large numbers of registers (such as the Alpha21264) than the 128-bit block ciphering in AES (Rijndael)[12]. The reason for this is that MISTY1 is extremely compact in hardware.

IMPLEMENTATION ON MICROCOMPUTERS. Tables 3 and 4 show the results of implementing MISTY1 on microcomputers included in other equipment. Here, the targets were the Mitsubishi Electric Corporation M32R, M16C, and Z80 microcomputers. The M32R and M16C are respectively 32-bit and 16-bit microcomputers developed independently by the corporation. These are general-purpose microcomputers that feature full product lines and are used broadly in a wide range of fields. Although at this point we show the results of implementation using code developed with an emphasis on speed, the cipher can also be implemented with an emphasis on reducing the size of the code. The Z80 is a standard environment for evaluating the implementation of symmetric-key ciphers on 8-bit microcomputers. It is desirable to use only a small amount of RAM in an 8-bit microcomputer. In an implementation where the emphasis is placed on reducing the amount of RAM used, the MISTY1 cipher was successfully miniaturized and accelerated.

Table 3 Mitsubishi Electric Microcomputerrs

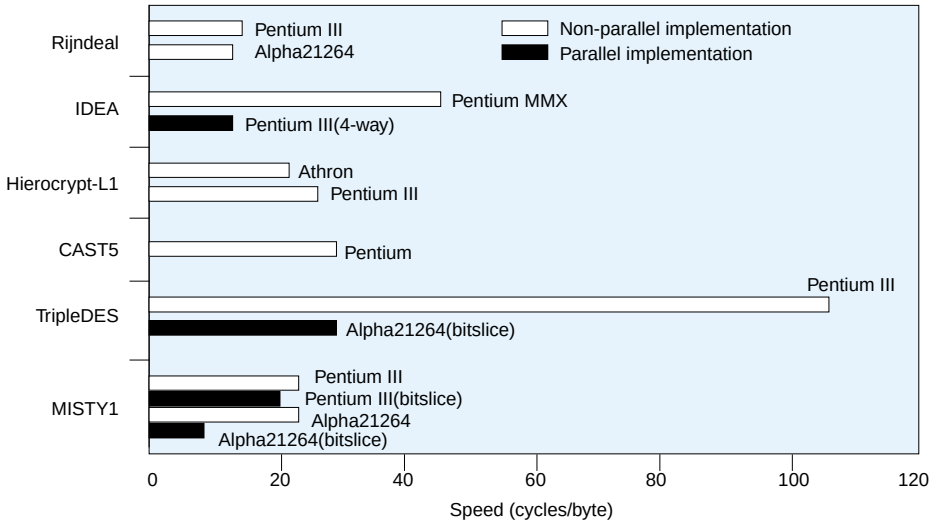
Micro-processor (Mhz)	Expanded key generation (cycles/key)	Cipher speed (cycles/block)	ROM/RAM (kbytes / bytes)
M16C(20)	743	1877	3.4 / 64
M32R(100)	387	793	3.7 / 76

Table 4 Z80

Expanded key generation	Cipher speed*	ROM (code + S-box)	RAM (expanded key/stack)
3283 States	13553 States	1992-byte	64-byte/ 10-byte

Implementation in Hardware

Two aspects must be considered in implementing cipher algorithms in hardware. The first concerns incorporation into applications. This must be done under the conditions required by the application, and thus must be optimized for each application. The second concerns evaluating the implementation. Here the implementation is done while keeping in mind acceleration and miniaturization under fair evaluation conditions.



Source : Rijndeal[2], IDEA[3], Hierocrypt-L1[10], CAST5[3], TripleDES (Mitsubishi Electric), MISTY1[1]

Fig. 1 Ciphering speed (cycles/byte)

DESIGN ARCHITECTURE. Many cipher algorithms, such as MISTY[1], KASUMI[8], Camellia[9], and AES (Rijndael)[11] have fundamental functions that are recursive. The architecture of these types of algorithms can be implemented using the following structures as shown in Fig. 2.

1. Fully loop-unrolled architecture
2. Loop architecture
3. Pipeline architecture

An example of a fully loop-unrolled implementation of an algorithm where an *F* function is repeated *n* times is given in Fig. 2a. In this architecture, all of the rounds of the function are implemented independently, where all calculations are performed in a single clock cycle. While this makes for rather large circuits, there are no registers that contain loop-structure selectors or intermediate values, and thus there is no delay-time overhead, so this approach produces high throughput.

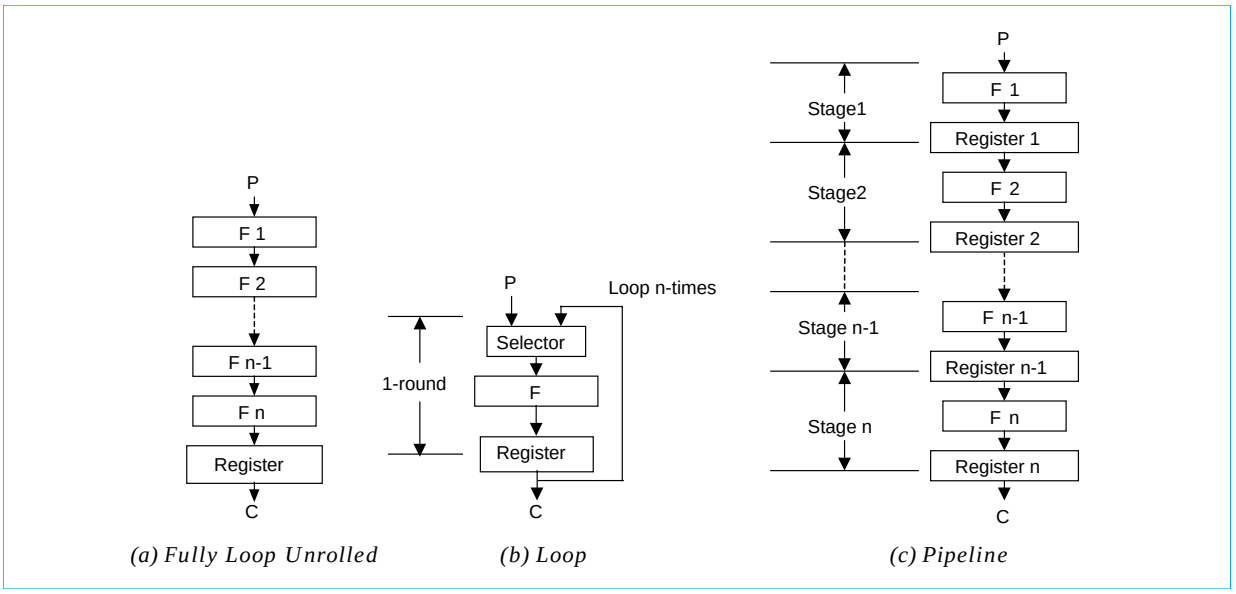
Fig. 2b shows an example of a loop architecture using only a single implementation of the *F* function in the same algorithm as in Fig. 2a. This architecture deploys only the essential function, and the processing of the entire algorithm is done by repeating the calculations. As a result, the circuit is small. On the other hand, because the processing of the cipher must be looped a number of times, there is increased overhead in the selectors that structure the loop and in the delay times in the registers, and so the throughput is impaired. The size of the circuit, the number of loops, and the throughput are determined by the selection of the fundamental calculation circuit that comprises each loop.

Fig. 2c is an example of an *n*-stage pipeline implementation of the same algorithm shown in Fig. 2a. This architecture divides up the entire algorithm into operating stages for each functional block, in a structure where each operating stage operates independently of the others. Because of this, the circuit is larger (by the amount of the registers) than the size of the circuit for the fully loop-unrolled architecture, but this architecture is also extremely fast because each of the stages can be performed in parallel. However, caution is required because in this architecture it is difficult to implement any loop-back processes in the cipher algorithm itself.

The FIPS (Federal Information Processing Standard) cipher-use modes[12]&[13] are often used in cipher algorithms, where there is a mode in which the data process in the previous circuit is chained to the next process (i.e., the cipher-block chaining (CBC), output feedback (OFB), and cipher feedback (CFB) modes) and also a mode wherein this chaining is not performed (i.e., electronic codebook (ECB) and counter (CTR)).

With CBC, OFB, and CFB, the mode in the first type of architecture (above) is appropriate in terms of the throughput of the architecture. For ECB and CTR, the third type of architecture is appropriate. On the other hand, the second type can be applied to all modes, and is also applicable in implementations where the emphasis is placed on the size of the circuit.

CONSIDERATIONS IN IMPLEMENTATIONS. Until now, we have implemented into ASIC (Application Specific Integrated Circuit) and FPGA (Field Programmable Gate Array) modular exponential arithmetic, which is usually used in RSA, and a



(a) Fully Loop Unrolled *(b) Loop* *(c) Pipeline*

Fig. 2 Hardware implementation architectures

variety of symmetric-key cipher algorithms including MISTY1 and Camellia.[14]~[18] In particular, for the FPGA the implementation was done using the Virtex series FPGA, produced by Xilinx Inc., which is often used in evaluations outside Japan.

In recent years, implementations in hardware have often used a hardware description language such as VHDL, Verilog-HDL, or the like, and as described in references [14] and [19], there may be differences in the performance in the hardware due to differences in the description method and in the method by which the design is actually implemented. Furthermore, the type of description should be modified depending on the specifics of the device targeted for the implementation (i.e., depending on whether it will be in an ASIC or in an FPGA) in order to increase its efficiency.

It is also important to look at the tradeoffs between circuit size and throughput when selecting the architecture to be used.

When implementing cipher algorithms in hardware, a knowledge, for example, of mathematics is required in addition to a knowledge of hardware design engineering. For example, in symmetric-key cipher algorithms, a non-linear replacement table (SBOX) is often used, and in recent years the number of these SBOX implementations based on Galois field calculations has been on the rise. Because of this, knowledge of mathematics is critical in reducing the size of the hardware, and knowledge of the Galois fields and logic-compression techniques are essential in reducing the size of the SBOX.

As described above, a variety of technologies, such as those for understanding the tradeoffs between circuit size and throughput, a knowledge of cipher algorithms and of mathematics, a knowledge of device characteristics and logic-synthesis technologies are all required in addition to hardware-design engineering in order to implement cipher algorithms efficiently.

Intellectual Property Regarding Cipher Algorithms for Security IC Development

Functional blocks that can be reused in the development of ASICs and FPGAs are known as intellectual properties (IPs). Advances in semiconductor technology have made the development of large-scale ICs possible, and the reuse of intellectual properties has vastly increased the efficiency of the design process.

On the other hand, in the networked society of today, ICs with ciphering functions are seen everywhere, used with the Internet, mobile telephones, DVDs, IC cards and in a host of other places. Intellectual properties for cipher algorithms for the development of security ICs (referred to simply as "cryptographic IPs" below) have been developed as a method by which to expedite the development of ICs with security functions and to apply to a variety of products the implementation technologies described above.

Types of IP

SOFT IP. A soft IP is one provided in the form of HDL (Hardware Description Language) from which logic circuits can be synthesized. This type of intellectual property is extremely flexible, and can be applied to a variety of IC manufacturing processes.

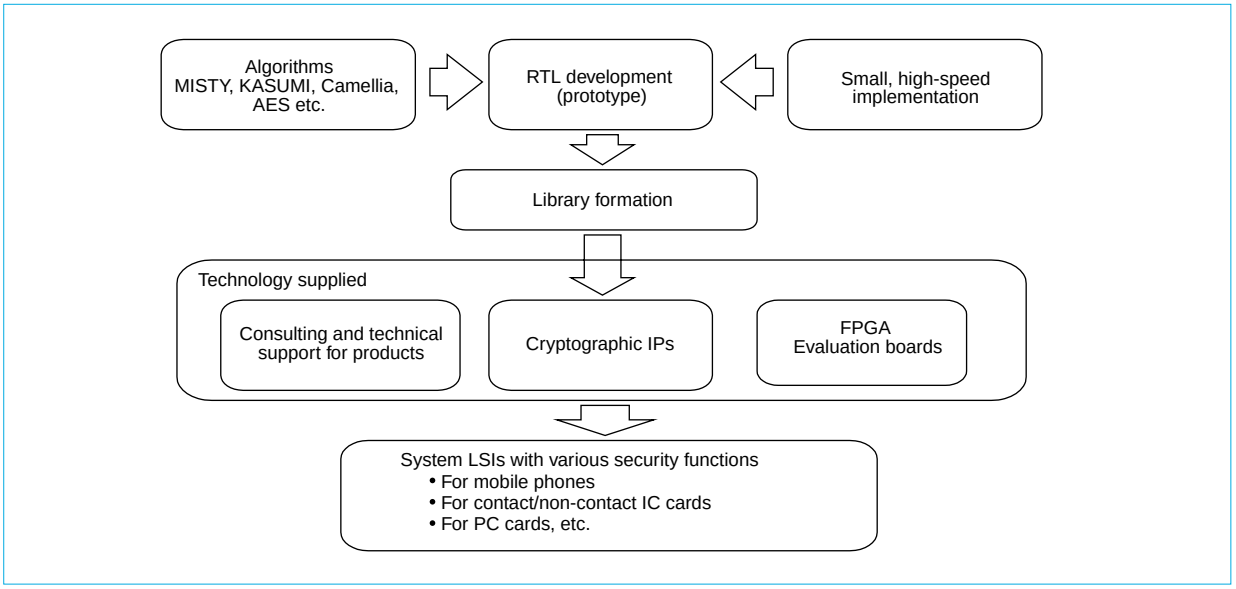


Fig. 3 Positioning of cryptographic IPs

FIRM IP. A firm IP is one provided in the form of logic-synthesis HDL, IC manufacturing process libraries, floor plan information, or net lists.

HARD IP. A hard IP is one provided in the form of layout and interconnect data for a specific manufacturing process. It is optimized for the particular process, and thus lacks flexibility in use.

The cryptographic IPs we have developed fall into the category of soft IP described above. However, because the HDL for logic circuit synthesis, which is the form in which soft IPs are usually provided, is normally readable, there has been no way to prevent leakage of the impementation technology through reverse engineering of the HDL. Additionally, because it can be assumed that the sophisticated implementation technology and know-how will be disclosed when the soft IP is provided, soft IPs have only been available at extremely high cost. We have developed our cryptographic IPs as a design compiler library for Synopsys Inc., which has become the *de facto* standard logic synthesis tool for IC development in the industry. The use of this method has made it possible to provide inexpensive IPs with all of the flexibility of a soft IP but without the leakage of implementation know-how.

Each of our cryptographic IPs includes a license-control function, and thus is able to protect this know-how, see Fig. 3.

Table 5 shows the detail of what is provided as the cryptographic IP, and Table 6 shows the supported cipher algorithms. Not only does this approach support the most up-to-date cipher algorithms but it also features the fact that it can include the provision of customized technologies and of development environments.

Table 5 Descriptions of Cryptographic IP Offerings

Item	Description
Logical synthesis tool library	Standard Evaluate (only)
Test vector	Verilog-HDL operating model VHDL operating model PLI model
Sample description	Verilog-HDL/VHDL Logical synthesis script
Service, etc.	Version updates I/F circuit customization FPGA board, etc., development environment

Table 6 Timing of Cryptographic IP Offerings

First available	Supported algorithms
From 2nd quarter 2001	Camellia, MISTY, KASUMI
From 3rd quarter 2001	AES, RSA accelerator
From 4th quarter 2001	SHA-1
From 3rd quarter 2002	SHA-192, 256, 384

The article summarizes some of the important considerations and options available to ensure the optimum implementation of ciphers. The establishment of evaluation methods for the ability to implement cipher algorithms, compatible with the most up-to-date examples, remains as an urgent priority for future development. ❑

References:

[1] Matsui, M.: New Encryption Algorithm MISTY. Proceedings of the Fourth International Workshop of Fast Software Encryption. 54~86 (1997-1)

[2] Junko Nakajima, et al: Optimized Software Deployment of the Symmetric-Key MISTY1, SCIS 2001 (2001-1)

[3] Nakajima, J., et al: Fast Software Implementation of MISTY1 on Alpha Processors, IEICE Transaction, E82-A, No.1, 107~116 (1999-1)

[4] Junko Nakajima, et al: Method for High-Speed Deployment of MISTY in Software (II), SCIS 98-9.1.B (1998-1)

[5] Matsui, M, et al: Method for High-Speed Deployment of MISTY in Software (III), ISEC 2000 (2000-11)

[6] Weoss, R.: A comparison of AES candidates on the Alpha 21264, Proceedings of the Third Advanced Encryption Standard Candidate Conference (2000-4)

[7] Lipmaa, H.: Fast IDEA for Pentium MMX compatibles, http://www.tml.hut.fi/~helger/fastidea/

[8] ETSI/SAGE. KASUMI specification: Specification of the 3GPP Confidentiality and Integrity Algorithms Document 2, ETSI/SAGE (1999-12)

[9] Kazumaro Aoki, et al: 128 Bit-Block Ciphers, Technical Report of the Institute of Electronics, Information and and Communications Engineers, ISEC 2000-6 (2000-5)

[10] Fumihiko Sano: Deployment of Hierocrypt, SCIS 2001 (2001-1)

[11] Advanced Encryption Standard, CFIPS 197 (2001-11)

[12] DES Modes of Operation, FIPS 81 (1980-12)

[13] Recommendation for Block Cipher Modes of Operation-Methods and Techniques, SP 800-38A (2001-12)

[14] Tetsuya Ichikawa, et al: Hardware Design of Private-Key Ciphers, Symposium on Ciphers and Information Security 1997, SCIS 97-9D (1997-1)

[15] Ichikawa, T., et al: Hardware Evaluation of the AES Finalists, in The Third AES Candidate Conference, printed by the National Institute of Standards and Technology, Gaithersburg, MD, 279~285 (2000-4)

[16] Tetsuya Ichikawa, et al: Hardware Deployment of a 128-Bit Block Cipher (III), Symposium on Ciphers and Information Security 2001 Preprints, SCIS 2001-12A-5, 669~674 (2001-1)

[17] Toru Sorimachi, et al: Evaluation Metrics for Hardware Deployments of Block Ciphers (2), Information Science and Engineering Communications, ISEC 2001-54 (2001-9)

[18] Shigeo Hasegawa, et al: Distributed Processing of Exponential Modulus Arithmetic Circuits for Use in Redundant Binary Expressions, Computer Architecture Research Committee 98-ARC-129-8 (1998-5)

[19] Satoh, A., et al: A Compact Rijndael Hardware Architecture with S-Box Optimization, Advances in Cryptology - ASIACRYPT 2001, LNCS 2248, 239~254 (2001)

Quantum Cryptography

*by Toshio Hasegawa and Tsuyoshi Nishioka**

Quantum cryptography is a new encryption cryptographic technology that stands at the confluence [fusion] of computer science and physics[1]. In contrast with modern cryptography, where security is evaluated on the basis of computational complexities (the amount of time required to break the encryption), quantum cryptography provides absolute security because it is based on fundamental physical laws. This article describes these fundamental principles of quantum cryptography technology, a method to implement it, and the data processes involved. It also describes the quantum-encryption communications system experiments that have been performed successfully by Mitsubishi Electric in cooperation with Hokkaido University.

What Is Quantum Cryptography?

Quantum-information technology, born of the fusion of information science and quantum mechanics, has been the subject of considerable attention since the proposal of P. Shor's high-speed quantum algorithm for factoralization. This is because the technology could conceivably make it easy to break modern ciphers in which security is based on a computational complexity theory, such as public-key cryptosystems. While, in terms of information-security technology, this type of quantum information technology provides a powerful attack tool in the form of quantum computers, it also provides an equally powerful defensive shield in terms of quantum cryptography. This shield is an encryption technology based on an entirely new concept of security-quantum-mechanical security-rather than depending on security based on computational complexity. Therefore, in quantum cryptography absolute security is guaranteed even with the advent of ultrahigh-performance computers that would inevitably pose a threat to modern cryptography. Furthermore, quantum cryptography holds the promise of "ultimate security" because it will be provided with new features, such as eavesdropping detection.

Here, we will explain a quantum key-distribution protocol that has been the most successful form of quantum cryptography in an actual application. This protocol allows two users (Alice and Bob) to share secret keys over a long distance with absolute security.

Fundamental Principles of Quantum Cryptography

Quantum cryptography differs from conventional optical communication in that single quantum particles (photons) are transmitted with each item of information Because of this, the quantum aspect of photons–the uncertainty principle and the no-cloning theorem–comes into play. The no-cloning theorem states that it is impossible to create a perfect copy of an unknown quantum state. Additionally, according to the uncertainty principle, there are certain conjugate physical quantities that cannot be accurately observed simultaneously. In other words, if one physical quantity is observed accurately, any observation of the other physical quantity becomes meaningless. These fundamental laws make it impossible to copy a quantum state with information (the so-called "qubit"), with the laws of physics guaranteeing that, once a quantum state has been observed, then if the state is observed again, one observation will be incorrect. This leads to so-called "once-and-for-all data transmission."

Quantum key-distribution protocols use these principles. The BB84 protocol, the best known of them, is explained below (with reference to Fig. 1).

Alice prepares four different states for one qubit. Of these four states, two form pairs, for instance horizontal or vertical polarization and left- or right-circular polarization. Each pair corresponds to one physical quantity, so that if an attempt is made to measure one of the pairs correctly, it becomes impossible to measure the other pair correctly. Bob has two kinds of receiver (measuring instrument) corresponding to the two pairs, and is able to measure the state of a pair correctly only when the correct instrument is selected. If Bob attempts to remeasure because he has selected the wrong instrument, the physical laws described above will make it impossible for him to do so. The key-sharing procedure is as described below:

1. Alice and Bob independently prepare random numbers. For each individual qubit, Alice selects one of the four states using random numbers to make her choice random, and sends it to Bob through the quantum communication

**Toshio Hasegawa and Tsuyoshi Nishioka are with the Information Technology R&D Center.*

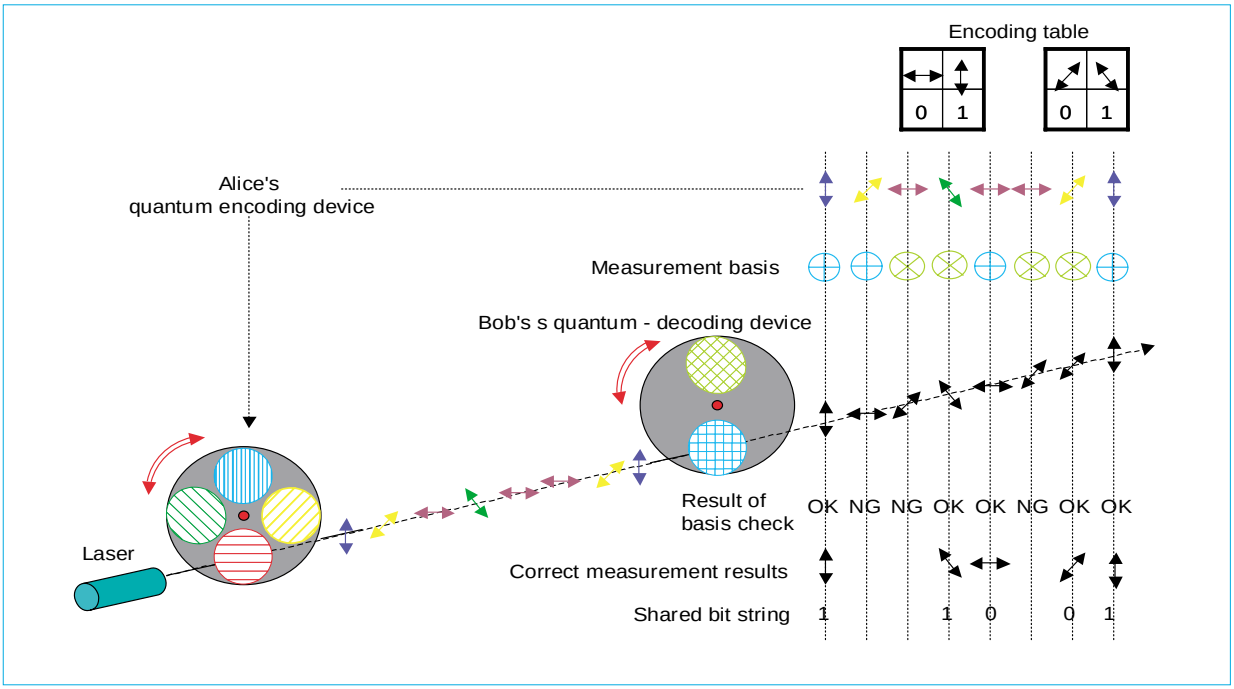


Fig. 1 BB84 protocol (for key distribution)

channel. Bob, according to his own random numbers, randomly picks one of the two measuring instruments to measure the transmitted qubits. The measurement results are recorded secretly.

2. Once the qubits have been received, Bob tells Alice which measuring instrument he used for each measurement, and Alice tells Bob which of these measurements was made with the appropriate instrument. This can be done over a public communication channel (ie, one where there is the risk of eavesdropping).
3. Next, Alice and Bob extract only those qubits that were measured correctly, so the two now secretly share a series of qubits. When this is converted into a series of digital bits, they secretly share an identical bit stream.

In an actual environment, a few of the shared data may actually not be identical due to noise or eavesdropping. This necessitates data processing peculiar to quantum cryptography as described below. In this data-processing procedure, bit-error correction (elimination) and privacy amplification are performed in order to reduce as far as possible data tapped by an eavesdropper. Because both the bit rate and the quantum-bit error rate (QBER) are monitored and estimated, any disruption by an eavesdropper would be detected, and so eavesdropping is detectable.

IMPLEMENTATION. When implementing quantum cryptography it is possible, in principle, to use a variety of quantum systems with two states; in experiments so far, photons have been used to carry information. This is because photons are easy to handle and there is a long history of successful optical communications.

There are two main methods of encoding data into photons; polarization encoding and phase encoding.The transmission channel for photons can be either air or optical fiber. For the key-distribution protocol, there are BB84, B92, E91 and several others. Therefore, there are a number of ways to implement it. Several experiments in quantum cryptography have already been performed by IBM, the University of Geneva and the Los Alamos National Laboratory in the United States. We have also performed such experiments. These have typically adopted phase encoding with transmission through an optical fiber. In the sections below we will explain the generation of the photons, their transmission, and their detection.

GENERATION OF PHOTONS. The carrier of information for quantum cryptography is a single photon. At present, the typical case is one in which the light emitted from a pulse laser is reduced using an optical attenuator until essentially unity is achieved. For example, if we set the average number of photons per pulse as 0.1, then the probability of there being more than one photon per pulse becomes essentially negligible. We can eliminate this effect by performing the data processing described below. However, if a single-photon generating technology were established, it would be possible to achieve quantum cryptography at a higher speed than possible today, and because this technology is also essential for other aspects of quantum information technology, the single-photon generator is an important research theme.

TRANSMISSION OF PHOTONS. Although there are two types of transmission (free-space and optical-fiber transmission), at present experiments tend to focus on the use of optical fibers because of the relative ease in maintaining the stability of the system. In optical-fiber transmission, phase encoding is used for its simplicity. This encoding often uses, for example, Mach-Zehnder interferometry. In this case, one must carefully consider the structure to ensure stability because the polarization of the photons is affected by birefringence, polarity dispersion, etc. When it comes to the wavelength, while both short and long wavelengths are available, long wavelengths are more suitable for long-distance transmissions in terms of transmission losses.

In free-space transmission there have recently been experiments with satellite communications using short wavelengths, increasing the potential applications for this technology.

DETECTION OF PHOTONS. For the detection of single photons, at present the best method is the use of an avalanche photodiode (APD). Depending on the wavelengths, three different types of semiconductors (Si, Ge, or InGaAs) may be used. There are commercial detectors with high detection efficiency at room temperature for short wavelengths. However, experiments performed with, for example, InGaAs at the longer wavelengths (such as 1,550nm) reveal the defects of low detection efficiency. In addition, because reducing the temperature to, say, 77K for long wavelengths requires bulky equipment, there is still a need to develop an efficient APD optimized for quantum cryptography.

Data Processing Required for an Actual System

In an ideal environment, achieving the absolute security of quantum cryptography, it is possible to detect eavesdroppers when using a key-distribution protocol such as BB84. In an actual environment, there are inevitably physical errors. This makes it possible for Eve the Eavesdropper to tap a few fragments of information without detection by Alice or Bob under physical errors. Normally, error bits have a high probability of resulting from Eve's eavesdropping. Therefore, security can be enhanced by detecting and eliminating those error bits (i.e., through error correction). However, if the photon generator transmits not one but two photons per pulse (despite an extremely small probability that this will occur), Eve will be able to steal information by getting one of the photons without errors in the quantum communications, or in other words, without arousing the attention either Alice or Bob. So there will still exist a small probability of the leakage of partial information even if error bits are eliminated. The method where the information is mapped uniformly into a small space so that information cannot leak to Eve is "privacy amplification." Here, we describe the essential data-processing procedures in terms of some simple examples. The following is a simple example of error correction.

1. Alice selects a two-bit pair at random, and tells Bob the results of an exclusive OR operation (XOR) for it.
2. Bob tells Alice whether or not the XOR value of the corresponding bits was the same.
3. If the value is the same, then the first bit is kept from the bit pair, and the second bit is discarded.
4. If different, then both bits are discarded. Here, one has to discard the second bit, which is peculiar to quantum cryptography, in order to correct errors while keeping security. This

makes it possible to prevent single bits from leaking to Eve.[3]

The following is an explanation of privacy amplification

1. Alice again selects a two-bit pair at random, and calculates its XOR value. However, this time she does not disclose the value to Bob.
2. Alice announces only which bits she has selected.
3. Alice and Bob each replace the two bits with their XOR values. By doing this, if Eve were to learn only the first bit but not know the second bit, then she would learn nothing about their XOR value. If Eve were to know both bits with finite probability, still the probability that she could correctly infer the XOR value would be reduced, and thus the security has been amplified.

A Quantum Cryptosystem by Mitsubishi Electric Corporation

The following section describes an experimental quantum cryptosystem and experiments performed by Mitsubishi Electric.

EXPERIMENTAL SETUP. The experimental system, as shown in Fig. 2, consists of three subsystems: the optics system, the control system, and the data-processing system.

1. Optics System
The optics system comprises a pulse laser that transmits photons, an optical fiber that serves as the transmission channel, a photon detector that detects the photons, and so on. We adopted phase encoding, and both parties used phase modulators to assign information to a photon.

2. Electronic Control System
Each entity controls its phase modulator electronically to assign information to a photon with the correct timing.

3. Data-Processing System
A data-processing system generated digital data (0,1) on each PC, and transmitted this information to the electronic control system. This applied a voltage corresponding to the bit values it received to modulate the phase in the optics system. Then, after optical transmission, the quantum key distribution protocol was used so that both parties could share the same data (secret key).

Table 1 Features and Performance of Our Quantum Cryptosystem

Operating environment	Room temperatures
Functions	- Sharing secret key between two entities with absolute security. - Detection of eavesdropping - Error correction and privacy amplification
System performance	Transmission distance: 200m Key-sharing speed: 1.1kbps Error rate: 1.7% For 1km, these become 0.7kbps, 5.0%.

SYSTEM FEATURES AND PERFORMANCE. To implement a quantum cryptosystem as a total security system, we adopted BB84 as the key-sharing protocol, phase encoding as the encoding method, an optical fiber as the communication channel, and 830nm as the light wavelength. The functions and performance of our experimental system were as shown in Table 1. The results are significant as the first in Japan, not as experimental proof-of-concept but as the implementation of a total quantum cryptosystem including data-processing.[2][3] The transmission speed (key distribution speed) of 1.1 kbps, and the error rate of 1.7%, both represent high performance achievements of world class. As for transmission distances, the experiment was successful over 1km.[4] The speed obtained is in the practical range when one considers its application of sharing the security keys for cryptography. These experimental values (transmission distance, speed, and error rate) have not yet been optimized, and thus further performance improvements can be expected. In addition, because a new method has been suggested in the quantum-cryptography optics system, it has become possible to perform the key sharing at a speed approximately six times that of the current method, doing so using a simpler system, and there has also been success in proof-of-concept experiments for multi-party extensions to quantum cryptography.[5]

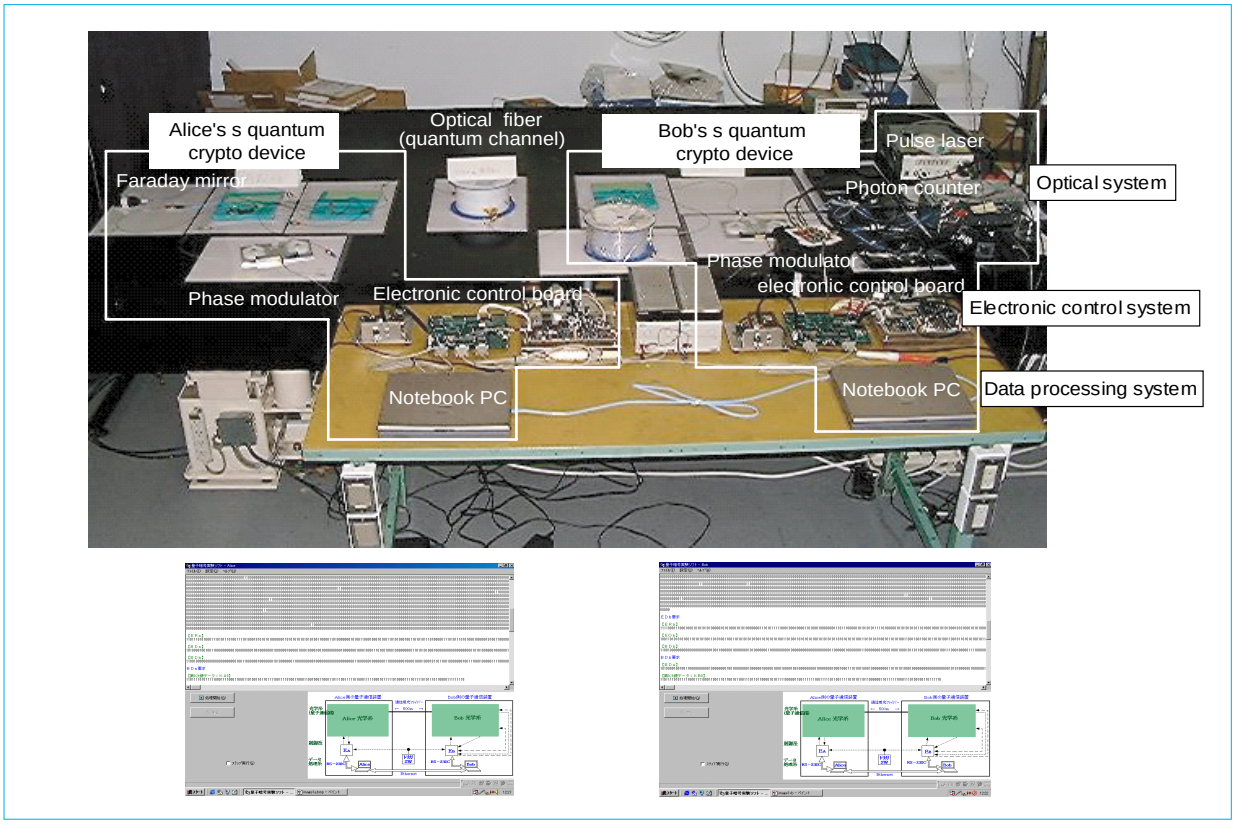


Fig. 2 Experimental setup of quantum cryptosystem

USER INTERFACE. As can be seen in Fig. 2, this experimental system featured a user interface facilitating operations on a personal computer.

This paper has provided a brief overview of quantum cryptography and has explained a practical method of achieving it. It has also described a successful quantum cryptosystem developed by the corporation and an experimental configuration and user interface on a personal computer. We further explained a practical quantum cryptosystem as a total security system. We also attempted to enhance the performance of a single-photon detector and system-control technology to make it more attractive as a commercial product. ❑

References

[1] Nishioka, T., et al: Quantum Cryptography - Theory and Experiments, Mathematical Sciences (9/2000)

[2] Mitsubishi Electric Corporate News Release, (9/14/2000)

[3] Hasegawa, T., et al: Experimental Quantum Cryptography, QIT 2000-42 (11/2000)

[4] Hasegawa, T., et al: An Experimental Realization of Quantum Cryptosystem, IEICE Trans. Fundamental, E85-A, No. 1. (2002)

[5] Nishioka, T., et al: Circular Type Quantum Key Distribution, IEEE Photonic Technology Letters Vol.14, No.4 (2002)

TURBOMISTY: A Tamper-Resistant Secure Board

*by Tetsuo Nakakawaji and Akira Takehara**

There are limits to what can be done in software to ensure the secure control of private-key information, central to the safety of any security system. Mitsubishi Electric has developed specialty hardware to provide a secure private key-control function using a tamper-resistant unauthorized-access prevention structure, where the specialty hardware is equipped with a load-balancing function using multiple boards.

Development Goals

TURBOMISTY was developed primarily to provide secure control of private keys. The United States Government has provided the FIPS140-1 standard as the basis for security in encryption modules, and recently there have been more cases requiring the use of tamper-resistant hardware complying with this standard. Given this, TURBOMISTY was designed in compliance with FIPS140-1, Level 3.

The purpose of the encryption board is not just to control the private keys but also to provide high-performance symmetric ciphers such as TripleDES and MISTY to be used as an encryption engine.

In the balance between cost and performance, the objective was not simply to provide a single board with the highest possible performance; instead, the development project improved the total performance level by load balancing among multiple boards.

Features

Fig. 1 shows an outside view of the TURBOMISTY. It provides the following features:

1. The TURBOMISTY provides a high-level of security using a tamper-resistant function in compliance with FIPS140-1, Level 3.
   - Physical protection of secure information such as key information and authorization parameters using hardware.
   - Automatic protection of illegal access to the hardware itself, with automatic erasure of the information contained therein.
2. A software interface complying with the PKCS (Public Key Cryptography Standards) #11, which has become the industry standard, as the key control API, providing interoperability with PKCS #11 applications from other companies.
3. The use of multiple boards makes distributed load balancing possible using multiple boards that are transparent to the application.
4. Because the TURBOMISTY is installed in the form of PCI boards, it can be moved relatively easily to another platform.
5. Other encryption algorithms, such as elliptic-curve encryption, can be added at later dates using firmware updates.

Functions

TAMPER-RESISTANT FUNCTIONS. Attempts to make the forms of illegal access listed below will be detected automatically, resulting in the automatic erasure of all secret information contained within the TURBOMISTY.

- Removal of the board from the host equipment.
- Removal of the cover from the board, or destruction thereof.

Fig. 1 External appearance of the TURBOMISTY

**Tetsuo Nakakawaji and Akira Takehara are with the Information Technology R&D Center.*

ASYMMETRIC CIPHER CALCULATIONS. RSA is used as the asymmetric cipher method, and, during calculations, all private-key information is concealed entirely within the board in the functions listed below in order to prevent any leakage to the outside:

- Public-key generation
- Private-key/public-key storage
- Private-key/public-key calculations
- Private-key backup

See below for more information about private-key backups.

SYMMETRIC CIPHER CALCULATIONS. Encryption and decryption calculation functions are provided using DES, Triple DES, and MISTY.

HARDWARE-BASED RANDOM NUMBER GENERATOR. It is possible to obtain better random numbers using hardware than it is with software-based generators.

DIGITAL CERTIFICATE STORAGE. This is based on X.509

MESSAGE DIGESTS. These are created using MD5 and SHA1.

SSL SERVER CONNECTIVITY. In the SSL protocol, which is used broadly as a secure communications protocol between browsers and web servers on the Internet, the control of the private keys for the SSL server is of critical importance. While at present most SSL servers store the private-key information on the disk drive, as is shown in Fig. 2, a combination of the TURBOMISTY and an SSL server makes it possible to control securely the private key information for the SSL server, used when establishing an SSL connection.

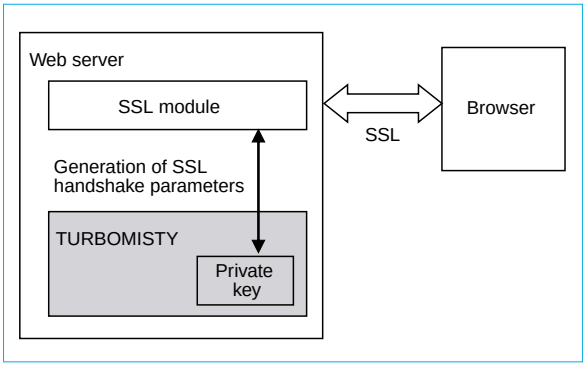


Fig. 2 Connections for an SSL server

The load on the CPU can also be reduced by performing in the TURBOMISTY the processes that are involved in encryption.

LOAD BALANCING. Load balancing among multiple boards is possible.

ADDITION OF NEW ALGORITHMS. Algorithms can be added by performing firmware updates for the TURBOMISTY.

Hardware Configuration

As is shown in Fig. 3, the TURBOMISTY is installed as a set of PCI boards. In order to prevent unauthorized access to key information or data from the circuits on the board during the calculations, the boards are structured with the circuits mounted on one side, with covers provided to cover the circuit-sides of the boards.

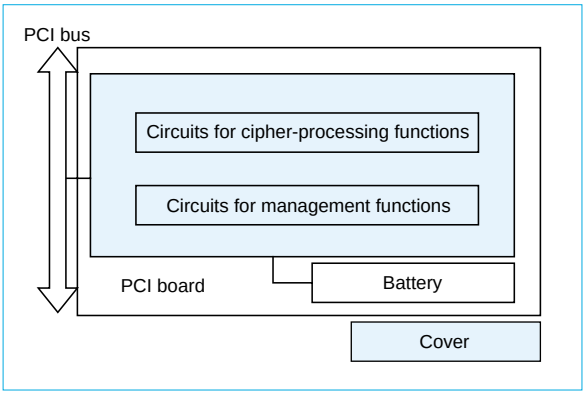


Fig. 3 Hardware configuration

A battery is provided outside the cover to maintain the key information when the host device is turned off, but all other components involved in the encryption processing and in managing keys are housed inside the cover.

Software Specifications

The PKCS #11, which has become the global standard for encryption module interfaces, is provided as the software interface.

In the TURBOMISTY, a single board has eight logical slots. So, for example, it can manage private keys from multiple certificate authorities (CAs). Furthermore, a maximum of four boards can be installed, meaning that up to 32 slots can be used. The relationships between the logi-

cal slots and the boards is managed within the PKCS #11 library.

As is shown in Fig. 4, non-exclusive control of the multiple ports that are fitted is performed within the PKCS #11 library, enabling processing to be distributed over multiple boards in a multi-thread/multi-processing environment without the need for any modifications to the applications.

Private Key Backups

TURBOMISTY's tamper-resistant mechanism prevents the leakage of key information by erasing automatically private keys stored within it. This function requires that the private keys be backed up against the possibility of a loss of security information due to attempted illegal access, damaged boards, etc., during operation.

Although the backup data is stored outside of the TURBOMISTY, the security is increased by internally encrypting the private keys and then dividing them up using a secret sharing procedure and then distributing components of the encrypted code to multiple users, thereby making it impossible to decode the original private key from the backup data outside of the TURBOMISTY.

In order to restore the private keys to the TURBOMISTY, the multiple backup segments that were indicated when the backup was made must be reassembled, preventing any individual from restoring the private keys working alone. The backup segments can be stored on floppy disks or IC cards. By storing the backup segments on IC cards, it is possible to prevent copying of the backup segments, making control even securer.

Performance

In addition to the advantage of security, the secure boards also provide enhanced performance. Conventionally, encryption processing and, in particular, private-key processing as part of an asymmetric cipher system, has consumed a large portion of CPU resources in multi-word calculations. By performing the encryption processes within the secure board, the load on the host CPU is reduced. The TURBOMISTY is able to perform RSA 1024-bit key signature calculations at a rate of six per second, and can perform RSA 2048-bit key signature calculations at a rate of about one per second.

In addition, when the signature generation is distributed among multiple boards, the performance increases essentially in direct proportion to the number of boards, thus making it possible to obtain a full load-balancing effect.

Managing Operations in the TURBOMISTY

The operation of the TURBOMISTY can be managed using the control tools on the host machine. The primary functions of the control tools are as follows:

- Board status display
- Board initialization/setup of parameters such as PINs

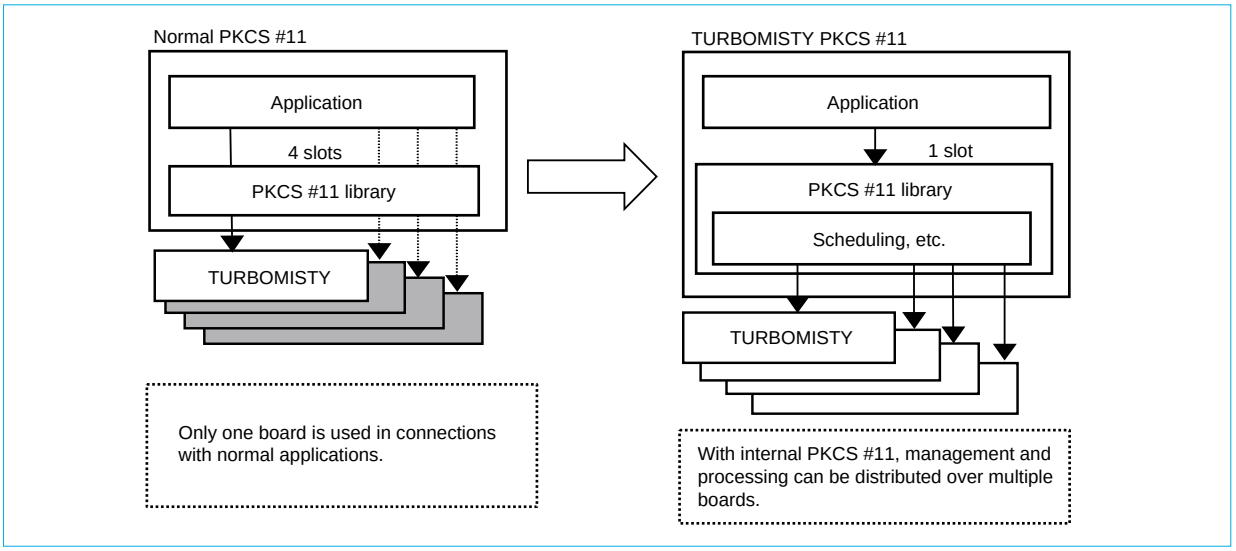


Fig. 4 Distributed processing using PKCS #11 library

- Display of lists of keys/digital certificates
- Backup/restore of private keys

Specifications

The primary specifications of the TURBOMISTY are given in Table 1.

Table 1 TURBOMISTY Specifications

PCI bus	32bit, 33MHz synchronous bus to PCI 2.1 standards
Size (W x Dmm)	Full-size PCI board 106.7 x 312
Compatible with FIPS 140-1	Level 3
Interface	PKCS #11 Ver. 2.01
Slots per board	8
Max. number of boards (slots)	4 (32)
Private keys per board	32
Certificates per board	64
Algorithms	RSA (512~2048) MISTY, DES, Triple DES MD5, SHA-1
Random number generator	Hardware
Compatible OS	Windows NT/2000 Solaris 7 HP-UX11.0 (planned)

Connectivity

At present, connectivity and interoperability of the TURBOMISTY has been verified for the following products:

ASSURETRANSACTION. AssureTransaction is the digital signature messaging system (DSMS) of Entegrity Corp., and is authorized as one component of the Identrus Financial System Certification System.

Connectivity between TURBOMISTY and AssureTransaction has been confirmed, certifying TURBOMISTY as an Entegrity Corp. certified hardware security module (HSM) along with products of foreign vendors such as Chrysalis Corp. and nCipher Corp.

IPLANET WEB SERVER. The iPlanet Web Server private key can be controlled by TURBOMISTY through storing them in the TURBOMISTY PKCS #11 library.

As electronic transaction systems and e-business become ubiquitous, hardware by which to control the keys that are central to the security of systems is expected to become increasingly important. The corporation is committed to remaining in the forefront of all such efforts. ❑

MITSUBISHI ELECTRIC CORPORATION
HEAD OFFICE: MITSUBISHI DENKI BLDG., MARUNOUCHI, TOKYO 100-8310, FAX 03-3218-3455