Network Working Group
Request for Comments: 3713
Category: Informational
M. Matsui
J. Nakajima
Mitsubishi Electric Corporation
S. Moriai
Sony Computer Entertainment Inc.
April 2004

A Description of the Camellia Encryption Algorithm

Status of this Memo

Copyright Notice

Abstract

This document describes the Camellia encryption algorithm. Camellia is a block cipher with 128-bit block size and 128-, 192-, and 256-bit keys. The algorithm description is presented together with key scheduling part and data randomizing part.

1. Introduction

1.1. Camellia

Camellia was jointly developed by Nippon Telegraph and Telephone Corporation and Mitsubishi Electric Corporation in 2000 [CamelliaSpec]. Camellia specifies the 128-bit block size and 128-, 192-, and 256-bit key sizes, the same interface as the Advanced Encryption Standard (AES). Camellia is characterized by its suitability for both software and hardware implementations as well as its high level of security. From a practical viewpoint, it is designed to enable flexibility in software and hardware implementations on 32-bit processors widely used over the Internet and many applications, 8-bit processors used in smart cards, cryptographic hardware, embedded systems, and so on [CamelliaTech]. Moreover, its key setup time is excellent, and its key agility is superior to that of AES.

Camellia has been scrutinized by the wide cryptographic community during several projects for evaluating crypto algorithms. In particular, Camellia was selected as a recommended cryptographic primitive by the EU NESSIE (New European Schemes for Signatures, Integrity and Encryption) project [NESSIE] and also included in the list of cryptographic techniques for Japanese e-Government systems which were selected by the Japan CRYPTREC (Cryptography Research and Evaluation Committees) [CRYPTREC].

2. Algorithm Description

Camellia can be divided into "key scheduling part" and "data randomizing part".

2.1. Terminology

The following operators are used in this document to describe the algorithm.

```
&    bitwise AND operation.
|    bitwise OR operation.
^    bitwise exclusive-OR operation.
<<   logical left shift operation.
>>   logical right shift operation.
<<<  left rotation operation.
~y   bitwise complement of y.
0x   hexadecimal representation.
```

Note that the logical left shift operation is done with the infinite data width.

The constant values of MASK8, MASK32, MASK64, and MASK128 are defined as follows.

```
MASK8   = 0xff;
MASK32  = 0xffffffff;
MASK64  = 0xffffffffffffffff;
MASK128 = 0xffffffffffffffffffffffffffffffff;
```

2.2. Key Scheduling Part

In the key schedule part of Camellia, the 128-bit variables of KL and KR are defined as follows. For 128-bit keys, the 128-bit key K is used as KL and KR is 0. For 192-bit keys, the leftmost 128-bits of key K are used as KL and the concatenation of the rightmost 64-bits of K and the complement of the rightmost 64-bits of K are used as KR. For 256-bit keys, the leftmost 128-bits of key K are used as KL and the rightmost 128-bits of K are used as KR.

```
128-bit key K:
    KL = K;    KR = 0;

192-bit key K:
    KL = K >> 64;
    KR = ((K & MASK64) << 64) | (~(K & MASK64));

256-bit key K:
    KL = K >> 128;
    KR = K & MASK128;
```

The 128-bit variables KA and KB are generated from KL and KR as follows. Note that KB is used only if the length of the secret key is 192 or 256 bits. D1 and D2 are 64-bit temporary variables. F-function is described in Section 2.4.

```
D1 = (KL ^ KR) >> 64;
D2 = (KL ^ KR) & MASK64;
D2 = D2 ^ F(D1, Sigma1);
D1 = D1 ^ F(D2, Sigma2);
D1 = D1 ^ (KL >> 64);
D2 = D2 ^ (KL & MASK64);
D2 = D2 ^ F(D1, Sigma3);
D1 = D1 ^ F(D2, Sigma4);
KA = (D1 << 64) | D2;
D1 = (KA ^ KR) >> 64;
D2 = (KA ^ KR) & MASK64;
D2 = D2 ^ F(D1, Sigma5);
D1 = D1 ^ F(D2, Sigma6);
KB = (D1 << 64) | D2;
```

The 64-bit constants Sigma1, Sigma2, ..., Sigma6 are used as "keys" in the F-function. These constant values are, in hexadecimal notation, as follows.

```
Sigma1 = 0xA09E667F3BCC908B;
Sigma2 = 0xB67AE8584CAA73B2;
Sigma3 = 0xC6EF372FE94F82BE;
Sigma4 = 0x54FF53A5F1D36F1C;
Sigma5 = 0x10E527FADE682D1D;
Sigma6 = 0xB05688C2B3E6C1FD;
```

64-bit subkeys are generated by rotating KL, KR, KA, and KB and taking the left- or right-half of them.

For 128-bit keys, 64-bit subkeys kw1, ..., kw4, k1, ..., k18, ke1, ..., ke4 are generated as follows.

```
kw1 = (KL <<<   0) >> 64;
kw2 = (KL <<<   0) & MASK64;
k1  = (KA <<<   0) >> 64;
k2  = (KA <<<   0) & MASK64;
k3  = (KL <<<  15) >> 64;
k4  = (KL <<<  15) & MASK64;
k5  = (KA <<<  15) >> 64;
k6  = (KA <<<  15) & MASK64;
ke1 = (KA <<<  30) >> 64;
ke2 = (KA <<<  30) & MASK64;
k7  = (KL <<<  45) >> 64;
k8  = (KL <<<  45) & MASK64;
k9  = (KA <<<  45) >> 64;
k10 = (KL <<<  60) & MASK64;
k11 = (KA <<<  60) >> 64;
k12 = (KA <<<  60) & MASK64;
ke3 = (KL <<<  77) >> 64;
ke4 = (KL <<<  77) & MASK64;
k13 = (KL <<<  94) >> 64;
k14 = (KL <<<  94) & MASK64;
k15 = (KA <<<  94) >> 64;
k16 = (KA <<<  94) & MASK64;
k17 = (KL <<< 111) >> 64;
k18 = (KL <<< 111) & MASK64;
kw3 = (KA <<< 111) >> 64;
kw4 = (KA <<< 111) & MASK64;
```

For 192- and 256-bit keys, 64-bit subkeys kw1, ..., kw4, k1, ..., k24, ke1, ..., ke6 are generated as follows.

```
kw1 = (KL <<<   0) >> 64;
kw2 = (KL <<<   0) & MASK64;
k1  = (KB <<<   0) >> 64;
k2  = (KB <<<   0) & MASK64;
k3  = (KR <<<  15) >> 64;
k4  = (KR <<<  15) & MASK64;
k5  = (KA <<<  15) >> 64;
k6  = (KA <<<  15) & MASK64;
ke1 = (KR <<<  30) >> 64;
ke2 = (KR <<<  30) & MASK64;
k7  = (KB <<<  30) >> 64;
k8  = (KB <<<  30) & MASK64;
k9  = (KL <<<  45) >> 64;
k10 = (KL <<<  45) & MASK64;
k11 = (KA <<<  45) >> 64;
```

```
k12 = (KA <<<  45) & MASK64;
ke3 = (KL <<<  60) >> 64;
ke4 = (KL <<<  60) & MASK64;
k13 = (KR <<<  60) >> 64;
k14 = (KR <<<  60) & MASK64;
k15 = (KB <<<  60) >> 64;
k16 = (KB <<<  60) & MASK64;
k17 = (KL <<<  77) >> 64;
k18 = (KL <<<  77) & MASK64;
ke5 = (KA <<<  77) >> 64;
ke6 = (KA <<<  77) & MASK64;
k19 = (KR <<<  94) >> 64;
k20 = (KR <<<  94) & MASK64;
k21 = (KA <<<  94) >> 64;
k22 = (KA <<<  94) & MASK64;
k23 = (KL <<< 111) >> 64;
k24 = (KL <<< 111) & MASK64;
kw3 = (KB <<< 111) >> 64;
kw4 = (KB <<< 111) & MASK64;
```

2.3. Data Randomizing Part

2.3.1. Encryption for 128-bit keys

128-bit plaintext M is divided into the left 64-bit D1 and the right 64-bit D2.

```
D1 = M >> 64;
D2 = M & MASK64;
```

Encryption is performed using an 18-round Feistel structure with FL- and FLINV-functions inserted every 6 rounds. F-function, FL-function, and FLINV-function are described in Section 2.4.

```
D1 = D1 ^ kw1;           // Prewhitening
D2 = D2 ^ kw2;
D2 = D2 ^ F(D1, k1);     // Round 1
D1 = D1 ^ F(D2, k2);     // Round 2
D2 = D2 ^ F(D1, k3);     // Round 3
D1 = D1 ^ F(D2, k4);     // Round 4
D2 = D2 ^ F(D1, k5);     // Round 5
D1 = D1 ^ F(D2, k6);     // Round 6
D1 = FL   (D1, ke1);     // FL
D2 = FLINV(D2, ke2);     // FLINV
D2 = D2 ^ F(D1, k7);     // Round 7
D1 = D1 ^ F(D2, k8);     // Round 8
D2 = D2 ^ F(D1, k9);     // Round 9
D1 = D1 ^ F(D2, k10);    // Round 10
```

```
D2 = D2 ^ F(D1, k11);     // Round 11
D1 = D1 ^ F(D2, k12);     // Round 12
D1 = FL   (D1, ke3);      // FL
D2 = FLINV(D2, ke4);      // FLINV
D2 = D2 ^ F(D1, k13);     // Round 13
D1 = D1 ^ F(D2, k14);     // Round 14
D2 = D2 ^ F(D1, k15);     // Round 15
D1 = D1 ^ F(D2, k16);     // Round 16
D2 = D2 ^ F(D1, k17);     // Round 17
D1 = D1 ^ F(D2, k18);     // Round 18
D2 = D2 ^ kw3;            // Postwhitening
D1 = D1 ^ kw4;
```

128-bit ciphertext C is constructed from D1 and D2 as follows.

```
C = (D2 << 64) | D1;
```

2.3.2. Encryption for 192- and 256-bit keys

128-bit plaintext M is divided into the left 64-bit D1 and the right 64-bit D2.

```
D1 = M >> 64;
D2 = M & MASK64;
```

Encryption is performed using a 24-round Feistel structure with FL- and FLINV-functions inserted every 6 rounds. F-function, FL-function, and FLINV-function are described in Section 2.4.

```
D1 = D1 ^ kw1;            // Prewhitening
D2 = D2 ^ kw2;
D2 = D2 ^ F(D1, k1);      // Round 1
D1 = D1 ^ F(D2, k2);      // Round 2
D2 = D2 ^ F(D1, k3);      // Round 3
D1 = D1 ^ F(D2, k4);      // Round 4
D2 = D2 ^ F(D1, k5);      // Round 5
D1 = D1 ^ F(D2, k6);      // Round 6
D1 = FL   (D1, ke1);      // FL
D2 = FLINV(D2, ke2);      // FLINV
D2 = D2 ^ F(D1, k7);      // Round 7
D1 = D1 ^ F(D2, k8);      // Round 8
D2 = D2 ^ F(D1, k9);      // Round 9
D1 = D1 ^ F(D2, k10);     // Round 10
D2 = D2 ^ F(D1, k11);     // Round 11
D1 = D1 ^ F(D2, k12);     // Round 12
D1 = FL   (D1, ke3);      // FL
D2 = FLINV(D2, ke4);      // FLINV
D2 = D2 ^ F(D1, k13);     // Round 13
```

```
D1 = D1 ^ F(D2, k14);     // Round 14
D2 = D2 ^ F(D1, k15);     // Round 15
D1 = D1 ^ F(D2, k16);     // Round 16
D2 = D2 ^ F(D1, k17);     // Round 17
D1 = D1 ^ F(D2, k18);     // Round 18
D1 = FL   (D1, ke5);      // FL
D2 = FLINV(D2, ke6);      // FLINV
D2 = D2 ^ F(D1, k19);     // Round 19
D1 = D1 ^ F(D2, k20);     // Round 20
D2 = D2 ^ F(D1, k21);     // Round 21
D1 = D1 ^ F(D2, k22);     // Round 22
D2 = D2 ^ F(D1, k23);     // Round 23
D1 = D1 ^ F(D2, k24);     // Round 24
D2 = D2 ^ kw3;            // Postwhitening
D1 = D1 ^ kw4;
```

128-bit ciphertext C is constructed from D1 and D2 as follows.

```
C = (D2 << 64) | D1;
```

2.3.3. Decryption

The decryption procedure of Camellia can be done in the same way as the encryption procedure by reversing the order of the subkeys.

That is to say:

```
128-bit key:
    kw1 <-> kw3
    kw2 <-> kw4
    k1  <-> k18
    k2  <-> k17
    k3  <-> k16
    k4  <-> k15
    k5  <-> k14
    k6  <-> k13
    k7  <-> k12
    k8  <-> k11
    k9  <-> k10
    ke1 <-> ke4
    ke2 <-> ke3

192- or 256-bit key:
    kw1 <-> kw3
    kw2 <-> kw4
    k1  <-> k24
    k2  <-> k23
    k3  <-> k22
```

```
k4  <-> k21
k5  <-> k20
k6  <-> k19
k7  <-> k18
k8  <-> k17
k9  <-> k16
k10 <-> k15
k11 <-> k14
k12 <-> k13
ke1 <-> ke6
ke2 <-> ke5
ke3 <-> ke4
```

2.4. Components of Camellia

2.4.1. F-function

F-function takes two parameters. One is 64-bit input data F_IN. The other is 64-bit subkey KE. F-function returns 64-bit data F_OUT.

```
F(F_IN, KE)
begin
    var x as 64-bit unsigned integer;
    var t1, t2, t3, t4, t5, t6, t7, t8 as 8-bit unsigned integer;
    var y1, y2, y3, y4, y5, y6, y7, y8 as 8-bit unsigned integer;
    x  = F_IN ^ KE;
    t1 =  x >> 56;
    t2 = (x >> 48) & MASK8;
    t3 = (x >> 40) & MASK8;
    t4 = (x >> 32) & MASK8;
    t5 = (x >> 24) & MASK8;
    t6 = (x >> 16) & MASK8;
    t7 = (x >>  8) & MASK8;
    t8 =  x        & MASK8;
    t1 = SBOX1[t1];
    t2 = SBOX2[t2];
    t3 = SBOX3[t3];
    t4 = SBOX4[t4];
    t5 = SBOX2[t5];
    t6 = SBOX3[t6];
    t7 = SBOX4[t7];
    t8 = SBOX1[t8];
    y1 = t1 ^ t3 ^ t4 ^ t6 ^ t7 ^ t8;
    y2 = t1 ^ t2 ^ t4 ^ t5 ^ t7 ^ t8;
    y3 = t1 ^ t2 ^ t3 ^ t5 ^ t6 ^ t8;
    y4 = t2 ^ t3 ^ t4 ^ t5 ^ t6 ^ t7;
    y5 = t1 ^ t2 ^ t6 ^ t7 ^ t8;
    y6 = t2 ^ t3 ^ t5 ^ t7 ^ t8;
```

```
        y7 = t3 ^ t4 ^ t5 ^ t6 ^ t8;
        y8 = t1 ^ t4 ^ t5 ^ t6 ^ t7;
        F_OUT = (y1 << 56) | (y2 << 48) | (y3 << 40) | (y4 << 32)
        | (y5 << 24) | (y6 << 16) | (y7 <<  8) | y8;
        return FO_OUT;
    end.
```

SBOX1, SBOX2, SBOX3, and SBOX4 are lookup tables with 8-bit input/output data. SBOX2, SBOX3, and SBOX4 are defined using SBOX1 as follows:

```
        SBOX2[x] = SBOX1[x] <<< 1;
        SBOX3[x] = SBOX1[x] <<< 7;
        SBOX4[x] = SBOX1[x <<< 1];
```

SBOX1 is defined by the following table. For example, SBOX1[0x3d] equals 86.

SBOX1:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
00:	112	130	44	236	179	39	192	229	228	133	87	53	234	12	174	65
10:	35	239	107	147	69	25	165	33	237	14	79	78	29	101	146	189
20:	134	184	175	143	124	235	31	206	62	48	220	95	94	197	11	26
30:	166	225	57	202	213	71	93	61	217	1	90	214	81	86	108	77
40:	139	13	154	102	251	204	176	45	116	18	43	32	240	177	132	153
50:	223	76	203	194	52	126	118	5	109	183	169	49	209	23	4	215
60:	20	88	58	97	222	27	17	28	50	15	156	22	83	24	242	34
70:	254	68	207	178	195	181	122	145	36	8	232	168	96	252	105	80
80:	170	208	160	125	161	137	98	151	84	91	30	149	224	255	100	210
90:	16	196	0	72	163	247	117	219	138	3	230	218	9	63	221	148
a0:	135	92	131	2	205	74	144	51	115	103	246	243	157	127	191	226
b0:	82	155	216	38	200	55	198	59	129	150	111	75	19	190	99	46
c0:	233	121	167	140	159	110	188	142	41	245	249	182	47	253	180	89
d0:	120	152	6	106	231	70	113	186	212	37	171	66	136	162	141	250
e0:	114	7	185	85	248	238	172	10	54	73	42	104	60	56	241	164
f0:	64	40	211	123	187	201	67	193	21	227	173	244	119	199	128	158

2.4.2. FL- and FLINV-functions

FL-function takes two parameters. One is 64-bit input data FL_IN. The other is 64-bit subkey KE. FL-function returns 64-bit data FL_OUT.

```
    FL(FL_IN, KE)
    begin
        var x1, x2 as 32-bit unsigned integer;
        var k1, k2 as 32-bit unsigned integer;
        x1 = FL_IN >> 32;
```

```
        x2 = FL_IN & MASK32;
        k1 = KE >> 32;
        k2 = KE & MASK32;
        x2 = x2 ^ ((x1 & k1) <<< 1);
        x1 = x1 ^ (x2 | k2);
        FL_OUT = (x1 << 32) | x2;
    end.
```

FLINV-function is the inverse function of the FL-function.

```
    FLINV(FLINV_IN, KE)
    begin
        var y1, y2 as 32-bit unsigned integer;
        var k1, k2 as 32-bit unsigned integer;
        y1 = FLINV_IN >> 32;
        y2 = FLINV_IN & MASK32;
        k1 = KE >> 32;
        k2 = KE & MASK32;
        y1 = y1 ^ (y2 | k2);
        y2 = y2 ^ ((y1 & k1) <<< 1);
        FLINV_OUT = (y1 << 32) | y2;
    end.
```

3. Object Identifiers

The Object Identifier for Camellia with 128-bit key in Cipher Block Chaining (CBC) mode is as follows:

```
      id-camellia128-cbc OBJECT IDENTIFIER ::=
          { iso(1) member-body(2) 392 200011 61 security(1)
            algorithm(1) symmetric-encryption-algorithm(1)
            camellia128-cbc(2) }
```

The Object Identifier for Camellia with 192-bit key in Cipher Block Chaining (CBC) mode is as follows:

```
      id-camellia192-cbc OBJECT IDENTIFIER ::=
          { iso(1) member-body(2) 392 200011 61 security(1)
            algorithm(1) symmetric-encryption-algorithm(1)
            camellia192-cbc(3) }
```

The Object Identifier for Camellia with 256-bit key in Cipher Block Chaining (CBC) mode is as follows:

```
      id-camellia256-cbc OBJECT IDENTIFIER ::=
          { iso(1) member-body(2) 392 200011 61 security(1)
            algorithm(1) symmetric-encryption-algorithm(1)
            camellia256-cbc(4) }
```

The above algorithms need Initialization Vector (IV). To determine the value of IV, the above algorithms take parameters as follows:

```
CamelliaCBCParameter ::= CamelliaIV  --  Initialization Vector

CamelliaIV ::= OCTET STRING (SIZE(16))
```

When these object identifiers are used, plaintext is padded before encryption according to RFC2315 [RFC2315].

4. Security Considerations

The recent advances in cryptanalytic techniques are remarkable. A quantitative evaluation of security against powerful cryptanalytic techniques such as differential cryptanalysis and linear cryptanalysis is considered to be essential in designing any new block cipher. We evaluated the security of Camellia by utilizing state-of-the-art cryptanalytic techniques. We confirmed that Camellia has no differential and linear characteristics that hold with probability more than 2^(-128), which means that it is extremely unlikely that differential and linear attacks will succeed against the full 18-round Camellia. Moreover, Camellia was designed to offer security against other advanced cryptanalytic attacks including higher order differential attacks, interpolation attacks, related-key attacks, truncated differential attacks, and so on [Camellia].

5. Informative References

[CamelliaSpec] Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J. and T. Tokita, "Specification of Camellia --- a 128-bit Block Cipher". http://info.isl.ntt.co.jp/camellia/

[CamelliaTech] Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J. and T. Tokita, "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms". http://info.isl.ntt.co.jp/camellia/

[Camellia] Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J. and T. Tokita, "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis -", In Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, August 2000, Proceedings, Lecture Notes in Computer Science 2012, pp.39-56, Springer-Verlag, 2001.

[CRYPTREC] "CRYPTREC Advisory Committee Report FY2002", Ministry of Public Management, Home Affairs, Posts and Telecommunications, and Ministry of Economy, Trade and Industry, March 2003. http://www.soumu.go.jp/joho_tsusin/security/cryptrec.html, CRYPTREC home page by Information-technology Promotion Agency, Japan (IPA) http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html

[NESSIE] New European Schemes for Signatures, Integrity and Encryption (NESSIE) project. http://www.cryptonessie.org

[RFC2315] Kaliski, B., "PKCS #7: Cryptographic Message Syntax Version 1.5", RFC 2315, March 1998.

Appendix A. Example Data of Camellia

Here are test data for Camellia in hexadecimal form.

```
128-bit key
    Key       : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
    Plaintext : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
    Ciphertext: 67 67 31 38 54 96 69 73 08 57 06 56 48 ea be 43

192-bit key
    Key       : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
              : 00 11 22 33 44 55 66 77
    Plaintext : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
    Ciphertext: b4 99 34 01 b3 e9 96 f8 4e e5 ce e7 d7 9b 09 b9

256-bit key
    Key       : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
              : 00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
    Plaintext : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
    Ciphertext: 9a cc 23 7d ff 16 d7 6c 20 ef 7c 91 9e 3a 75 09
```

Acknowledgements

Shiho Moriai worked for NTT when this document was developed.

Authors' Addresses

Mitsuru Matsui
Mitsubishi Electric Corporation
Information Technology R&D Center
5-1-1 Ofuna, Kamakura
Kanagawa 247-8501, Japan

Phone: +81-467-41-2190
Fax: +81-467-41-2185
EMail: matsui@iss.isl.melco.co.jp

Junko Nakajima
Mitsubishi Electric Corporation
Information Technology R&D Center
5-1-1 Ofuna, Kamakura
Kanagawa 247-8501, Japan

Phone: +81-467-41-2190
Fax: +81-467-41-2185
EMail: june15@iss.isl.melco.co.jp

Shiho Moriai
Sony Computer Entertainment Inc.

Phone: +81-3-6438-7523
Fax: +81-3-6438-8629
EMail: shiho@rd.scei.sony.co.jp
camellia@isl.ntt.co.jp (Camellia team)

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.