Groupe de travail Réseau
Request for Comments : 3713
Catégorie : Information
Traduction Claude Brière de L'Isle

M. Matsui & J. Nakajima, Mitsubishi Electric Corporation
S. Moriai, Sony Computer Entertainment Inc.
avril 2004

Description de l'algorithme de chiffrement Camellia

Résumé
Le présent document décrit l'algorithme de chiffrement Camellia. Camellia est un chiffrement de bloc avec une taille de bloc de 128 bits et des clés de 128, 192, et 256 bits. La description de l'algorithme est présentée avec une partie de programmation de clé et une partie d'aléation des données.

1. Introduction

Camellia a été développé conjointement par Nippon Telegraph and Telephone Corporation et Mitsubishi Electric Corporation en 2000. Camellia spécifie la taille de bloc de 128 bits et les tailles de clé de 128, 192, et 256 bits, et la même interface que la norme de chiffrement évoluée (AES, *Advanced Encryption Standard*). Camellia se caractérise par sa convenance pour les mises en œuvre de logiciel et de matériel aussi bien que par son haut niveau de sécurité. D'un point de vue pratique, il est conçu pour apporter de la souplesse aux mises en œuvre de matériel et de logiciel sur les processeurs de 32 bits largement utilisés sur l'Internet et sur de nombreuses applications, sur les processeurs de 8 bits utilisés dans les cartes à mémoire, le matériel cryptographique, les systèmes incorporés, et ainsi de suite [CamelliaTech]. De plus, son délai d'établissement de clé est excellent, et son agilité de clé est supérieure à celle de AES.

Camellia a été examiné par toute la communauté de la cryptographie lors de plusieurs projets pour évaluer les algorithmes de chiffrement. En particulier, Camellia a été choisi comme primitive cryptographique recommandée par le projet de nouveaux schémas européens pour les signatures, l'intégrité et le chiffrement (NESSIE, *New European Schemes for Signatures, Integrity and Encryption*) [NESSIE] de l'Union Européenne et aussi inclus dans la liste des techniques de chiffrement pour les systèmes du gouvernement japonais qui ont été choisies par le comité de recherche et d'évaluation cryptographique (CRYPTREC, *Cryptography Research and Evaluation Committee*) [CRYPTREC] du Japon.

2. Description de l'algorithme

Camellia peut être divisé en "partie de programmation de clé" et "partie d'aléation des données".

2.1 Terminologie

Les opérateurs suivants sont utilisés dans le présent document pour décrire l'algorithme.

&	opération ET au bit près.
|	opération OU au bit près.
^	opération OU exclusive au bit près.
<<	opération logique de glissement à gauche.
>>	opération logique de glissement à droite.
<<<	opération de rotation à gauche.
~y	complément au bit près de y.
0x	représentation hexadécimale.

Noter que l'opération logique de glissement à gauche est faite avec la largeur de données infinie.

Les valeurs constantes de MASK8, MASK32, MASK64, et MASK128 sont définies comme suit :
MASK8 = 0xff;
MASK32 = 0xffffffff;
MASK64 = 0xffffffffffffffff;
MASK128 = 0xffffffffffffffffffffffffffffffff;

2.2 Partie programmation de clé

Dans la partie programmation de clé de Camellia, les variables de 128 bits de KL et de KR sont définies comme suit. Pour les clés de 128 bits, la clé K de 128 bits est utilisée comme KL et KR est 0. Pour les clés de 192 bits, les 128 bits de gauche de la clé K sont utilisés comme KL et l'enchaînement des 64 bits de droite de K et du complément des 64bits de droite de K est utilisé comme KR. Pour les clés de 256 bits, les 128 bits de gauche de la clé K sont utilisés comme KL et les 128 bits de droite de K sont utilisés comme KR.

Clé K de 128 bits : KL = K; KR = 0;

Clé K de 192 bits:
KL = K >> 64;
KR = ((K & MASK64) << 64) | (~(K & MASK64));

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

Les variables KA et KB de 128 bits sont générées à partir de KL et KR comme suit. Noter que KB n'est utilisé que si la longueur de la clé secrète est 192 ou 256 bits. D1 et D2 sont des variables temporaires de 64 bits. La fonction F- est décrite au paragraphe 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;

Les constantes de 64 bits Sigma1, Sigma2, ..., Sigma6 sont utilisées comme des "clés" dans la fonction F-. Ces valeurs constantes, en notation hexadécimale, sont :
Sigma1 = 0xA09E667F3BCC908B;
Sigma2 = 0xB67AE8584CAA73B2;
Sigma3 = 0xC6EF372FE94F82BE;
Sigma4 = 0x54FF53A5F1D36F1C;
Sigma5 = 0x10E527FADE682D1D;
Sigma6 = 0xB05688C2B3E6C1FD;

Les sous clés de 64 bits sont générées par rotation de KL, KR, KA, et KB et en en prenant la moitié gauche ou droite.

Pour les clés de 128 bits, les sous clés de 64 bits kw1, ..., kw4, k1, ..., k18, ke1, ..., ke4 sont générées comme suit :
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;

Pour les clés de 192 et de 256 bits, les ous clés de 64 bits kw1, .., kw4, k1, .., k24, ke1, .., ke6 sont générées comme suit :
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 Partie d'aléation des données

2.3.1 Chiffrement pour clés de 128 bits

Le texte source M de 128 bits est divisé en la moitié gauche D1 de 64 bits et la moitié droite D2 de 64 bits.
D1 = M >> 64;
D2 = M & MASK64;

Le chiffrement est effectué en utilisant une structure de Feistel à 18 tours avec les fonctions FL- et FLINV- insérées tous les 6 tours. Les fonctions F-, FL-, et FLINV- sont décrites au paragraphe 2.4.

```
D1 = D1 ^ kw1 ;          // Préblanchiment
D2 = D2 ^ kw2 ;
D2 = D2 ^ F(D1, k1) ;    // Tour 1
D1 = D1 ^ F(D2, k2) ;    // Tour 2
D2 = D2 ^ F(D1, k3) ;    // Tour 3
D1 = D1 ^ F(D2, k4) ;    // Tour 4
D2 = D2 ^ F(D1, k5) ;    // Tour 5
D1 = D1 ^ F(D2, k6) ;    // Tour 6
D1 = FL   (D1, ke1) ;    // FL
D2 = FLINV(D2, ke2) ;    // FLINV
D2 = D2 ^ F(D1, k7) ;    // Tour 7
D1 = D1 ^ F(D2, k8) ;    // Tour 8
D2 = D2 ^ F(D1, k9) ;    // Tour 9
D1 = D1 ^ F(D2, k10) ;   // Tour 10
D2 = D2 ^ F(D1, k11) ;   // Tour 11
D1 = D1 ^ F(D2, k12) ;   // Tour 12
D1 = FL   (D1, ke3) ;    // FL
D2 = FLINV(D2, ke4) ;    // FLINV
D2 = D2 ^ F(D1, k13) ;   // Tour 13
D1 = D1 ^ F(D2, k14) ;   // Tour 14
D2 = D2 ^ F(D1, k15) ;   // Tour 15
D1 = D1 ^ F(D2, k16) ;   // Tour 16
D2 = D2 ^ F(D1, k17) ;   // Tour 17
D1 = D1 ^ F(D2, k18) ;   // Tour 18
D2 = D2 ^ kw3;           // Postblanchiment
D1 = D1 ^ kw4.
```

Le texte chiffré C de 128 bits est construit à partir de D1 et D2 comme suit : C = (D2 << 64) | D1;

2.3.2 Chiffrement pour clés de 192 et 256 bits

Le texte source M de 128 bit est divisé en moitié de gauche D1 de 64 bits et de droite D2 de 64 bits :
D1 = M >> 64;
D2 = M & MASK64;

Le chiffrement est effectué en utilisant une structure de Feistel à 24 tours avec les fonctions FL- et FLINV- insérées tous les 6 tours. Les fonctions F-, FL-, et FLINV- sont décrites au paragraphe 2.4.

```
D1 = D1 ^ kw1 ;          // Préblanchiment
D2 = D2 ^ kw2 ;
D2 = D2 ^ F(D1, k1) ;    // Tour 1
D1 = D1 ^ F(D2, k2) ;    // Tour 2
D2 = D2 ^ F(D1, k3) ;    // Tour 3
D1 = D1 ^ F(D2, k4) ;    // Tour 4
D2 = D2 ^ F(D1, k5) ;    // Tour 5
D1 = D1 ^ F(D2, k6) ;    // Tour 6
D1 = FL   (D1, ke1) ;    // FL
D2 = FLINV(D2, ke2) ;    // FLINV
D2 = D2 ^ F(D1, k7) ;    // Tour 7
```

```
D1 = D1 ^ F(D2, k8) ;     // Tour 8
D2 = D2 ^ F(D1, k9) ;     // Tour 9
D1 = D1 ^ F(D2, k10) ;    // Tour 10
D2 = D2 ^ F(D1, k11) ;    // Tour 11
D1 = D1 ^ F(D2, k12) ;    // Tour 12
D1 = FL   (D1, ke3) ;     // FL
D2 = FLINV(D2, ke4) ;     // FLINV
D2 = D2 ^ F(D1, k13) ;    // Tour 13
D1 = D1 ^ F(D2, k14) ;    // Tour 14
D2 = D2 ^ F(D1, k15) ;    // Tour 15
D1 = D1 ^ F(D2, k16) ;    // Tour 16
D2 = D2 ^ F(D1, k17) ;    // Tour 17
D1 = D1 ^ F(D2, k18) ;    // Tour 18
D1 = FL   (D1, ke5) ;     // FL
D2 = FLINV(D2, ke6) ;     // FLINV
D2 = D2 ^ F(D1, k19) ;    // Tour 19
D1 = D1 ^ F(D2, k20) ;    // Tour 20
D2 = D2 ^ F(D1, k21);     // Tour 21
D1 = D1 ^ F(D2, k22) ;    // Tour 22
D2 = D2 ^ F(D1, k23) ;    // Tour 23
D1 = D1 ^ F(D2, k24) ;    // Tour 24
D2 = D2 ^ kw3 ;           // Postblanchiment
D1 = D1 ^ kw4.
```

Le texte chiffré C de 128 bits est construit à partir de D1 et D2 comme suit : C = (D2 << 64) | D1;

2.3.3 Déchiffrement

La procédure de déchiffrement de Camellia peut être faite de la même façon que la procédure de chiffrement en inversant l'ordre des sous clés.

C'est-à-dire :

Clé de 128 bits :
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

Clé de 192 ou 256 bits :
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 Composants de Camellia

2.4.1 Fonction F-

La fonction F- prend deux paramètres. L'un est l'entrée de 64 bits F_IN. L'autre est la sous clé de 64 bits KE. La fonction F- retourne les 64 bits de données de F_OUT.

```
F(F_IN, KE)
début
variable x comme entier non signé de 64 bits ;
variables t1, t2, t3, t4, t5, t6, t7, t8 comme entiers non signés de 8 bits ;
variables y1, y2, y3, y4, y5, y6, y7, y8 comme entiers non signés de 8 bits ;
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;
retourne FO_OUT;
Fin.
```

SBOX1, SBOX2, SBOX3, et SBOX4 sont des tableaux de recherche avec des données d'entrée/sortie de 8 bits. SBOX2, SBOX3, et SBOX4 sont définis en utilisant SBOX1 comme suit :

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

SBOX1 est défini par le tableau suivant. Par exemple, SBOX1[0x3d] égale 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	25	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 Fonctions FL- et FLINV-

La fonction FL- prend deux paramètres. L'un est l'entrée de données FL_IN de 64 bits, l'autre est la sous clé KE de 64 bits. La fonction FL- retourne les 64 bits de données FL_OUT.

```
FL(FL_IN, KE)
Début
   variables x1, x2 comme entiers non signés de 32 bits ;
   variables k1, k2 comme entiers non signés de 32 bits ;
   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;
Fin.
```

La fonction FLINV- est la fonction inverse de la fonction FL-.

```
FLINV(FLINV_IN, KE)
Début
   variables y1, y2 comme entiers non signés de 32 bits ;
   variables k1, k2 comme entiers non signés de 32 bits ;
   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;
Fin.
```

3. Identifiants d'objet

L'identifiant d'objet pour Camellia avec une clé de 128 bits en mode de chaînage de bloc de chiffrement (CBC) est le suivant :

IDENTIFIANT D'OBJET id-camellia128-cbc ::= { iso(1) member-body(2) 392 200011 61 security(1) algorithm(1) symmetric-encryption-algorithm(1) camellia128-cbc(2) }

L'identifiant d'objet pour Camellia avec une clé de 192 bits en mode de chaînage de bloc de chiffrement (CBC) est le suivant :

IDENTIFIANT D'OBJET id-camellia192-cbc ::= { iso(1) member-body(2) 392 200011 61 security(1) algorithm(1)
symmetric-encryption-algorithm(1) camellia192-cbc(3) }

L'identifiant d'objet pour Camellia avec une clé de 256 bits en mode de chaînage de bloc de chiffrement (CBC) est le suivant :

IDENTIFIANT D'OBJET id-camellia256-cbc ::= { iso(1) member-body(2) 392 200011 61 security(1) algorithm(1)
symmetric-encryption-algorithm(1) camellia256-cbc(4) }

Les algorithmes ci-dessus ont besoin d'une valeur d'initialisation (IV). Pour déterminr la valeur de l'IV, les algorithmes ci-dessus prennent des paramètres comme suit :

CamelliaCBCParameter ::= CamelliaIV -- Valeur d'initialisation

CamelliaIV ::= CHAÎNE D'OCTETS (TAILLE(16))

Quand ces identifiants d'objet sont utilisés, le texte source fait l'objet d'un bourrage avant le chiffrement conformément à la [RFC2315].

4. Considérations sur la sécurité

Les récentes avancées en techniques de cryptanalyse sont remarquables. Une évaluation quantitative de la sécurité contre les techniques puissantes de cryptanalyse telles que la cryptanalyse différentielle et la cryptanalyse linéaire est considérée comme essentielle dans la conception d'un nouveau chiffrement de bloc. On évalue la sécurité de Camellia en utilisant l'état de l'art des techniques de cryptanalyse. On a confirmé que Camellia n'a pas de caractéristiques différentielles et linéaires qui tiennent avec une probabilité de plus de 2^(-128), ce qui signifie qu'il est extrêmement improbable que des attaques différentielles et linéaires réussissent contre le Camellia complet à 18 tours. De plus, Camellia a été conçu pour offrir une sécurité contre les autres attaques avancées de cryptanalyse en incluant des attaques différentielles d'ordre supérieur, des attaques par interpolation, des attaques de relation de clés, des attaques différentielles tronquées, et ainsi de suite [Camellia].

5. Références pour information

[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., Morai, S., Nakajima, J. and T. Tokita, "Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis -", dans Selected Areas in Cryptography, 7th Annual International Workshop, SAC 2000, Waterloo, Ontario, Canada, août 2000, Proceedings, notes de lecture dans 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, mars 2003. http://www.soumu.go.jp/joho_tsusin/security/cryptrec.html ,
Page d'accueil CRYPTREC par l'agence de promotion des technologies de l'information du Japon (IPA) http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html

[NESSIE] Projet de nouveaux schémas européens pour les signatures, l'intégrité et le chiffrement (NESSIE). http://www.cryptonessie.org

[RFC2315] B. Kaliski, "PKCS n° 7 : Syntaxe de message cryptographique, version 1.5", mars 1998. *(Information)*

Appendice A. Exemple de données de Camellia

Voici des données d'essai pour Camellia en forme hexadécimale.

Clé de 128 bits
Clé : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Texte source : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Texte chiffré : 67 67 31 38 54 96 69 73 08 57 06 56 48 ea be 43

Clé de 192 bits
Clé : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
: 00 11 22 33 44 55 66 77
Texte source : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Texte chiffré : b4 99 34 01 b3 e9 96 f8 4e e5 ce e7 d7 9b 09 b9

Clé de 256 bits
Clé : 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
Clé de : 01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
Texte chiffré : 9a cc 23 7d ff 16 d7 6c 20 ef 7c 91 9e 3a 75 09

Remerciements

Shiho Moriai travaillait pour NTT quand ce document a été développé.

Adresse des auteurs

Mitsuru Matsui
Mitsubishi Electric Corporation
Information Technology R&D Center
5-1-1 Ofuna, Kamakura
Kanagawa 247-8501, Japan
téléphone : +81-467-41-2190
Fax: +81-467-41-2185
mél : 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
téléphone : +81-467-41-2190
Fax: +81-467-41-2185
mél : june15@iss.isl.melco.co.jp

Shiho Moriai
Sony Computer Entertainment Inc.
téléphone : +81-3-6438-7523
Fax: +81-3-6438-8629
mél : shiho@rd.scei.sony.co.jp
camellia@isl.ntt.co.jp (équipe Camellia)

Déclaration de droits de reproduction

mesure dans laquelle toute licence sur de tels droits pourrait être ou n'être pas disponible ; pas plus qu'elle ne prétend avoir accompli aucun effort pour identifier de tels droits. Les informations sur les procédures de l'ISOC au sujet des droits dans les documents de l'ISOC figurent dans les BCP 78 et BCP 79.

Des copies des dépôts d'IPR faites au secrétariat de l'IETF et toutes assurances de disponibilité de licences, ou le résultat de tentatives faites pour obtenir une licence ou permission générale d'utilisation de tels droits de propriété par ceux qui mettent en œuvre ou utilisent la présente spécification peuvent être obtenues sur répertoire en ligne des IPR de l'IETF à http://www.ietf.org/ipr .

L'IETF invite toute partie intéressée à porter son attention sur tous copyrights, licences ou applications de licence, ou autres droits de propriété qui pourraient couvrir les technologies qui peuvent être nécessaires pour mettre en œuvre la présente norme. Prière d'adresser les informations à l'IETF à ietf-ipr@ietf.org .

Remerciement
Le financement de la fonction d'édition des RFC est actuellement fourni par la Internet Society.