1. ВВЕДЕНИЕ

Шифрование - способ преобразования открытой информации в закрытую и обратно. Применяется для хранения важной информации в ненадёжных источниках или передачи её по незащищённым каналам связи. Согласно ГОСТ 28147-89, шифрование подразделяется на процесс зашифровывания и расшифровывания. В зависимости от алгоритма преобразования данных, методы шифрования подразделяются на гарантированной или временной криптостойкости. В зависимости от структуры используемых ключей методы шифрования подразделяются на:

• симметричное шифрование: посторонним лицам может быть известен алгоритм шифрования, но неизвестна небольшая порция секретной информации - ключа, одинакового для отправителя и получателя сообщения;

• асимметричное шифрование: посторонним лицам может быть известен алгоритм шифрования, и, возможно, открытый ключ, но неизвестен закрытый ключ, известный только получателю.

2. ИСТОКИ

В далеком 1998 году NIST объявил конкурс на создание алгоритма, удовлетворяющего выдвинутым институтом требованиям. Он опубликовал все несекретные данные о тестировании кандидатов на роль AES и потребовал от авторов алгоритмов сообщить о базовых принципах построения используемых в них констант. В отличие от ситуации с DES, NIST при выборе AES не стал опираться на секретные и, как следствие, запрещенные к публикации данные об исследовании алгоритмов-кандидатов. Чтобы быть утвержденным в качестве стандарта, алгоритм должен был:

1) реализовать шифрование частным ключом;

2) представлять собой блочный шифр;

3) работать со 128-разрядными блоками данных и ключами трех размеров (128, 192 и 256 разрядов).

Дополнительно кандидатам рекомендовалось:

1) использовать операции, легко реализуемые как аппаратно (в микрочипах), так и программно (на персональных компьютерах и серверах);

2) ориентироваться на 32-разрядные процессоры;

3) не усложнять без необходимости структуру шифра для того, чтобы все заинтересованные стороны были в состоянии самостоятельно провести независимый криптоанализ алгоритма и убедиться, что в нем не заложено каких-либо недокументированных возможностей.

Кроме того, алгоритм, претендующий на роль стандарта, должен распространяться по всему миру на неэксклюзивных условиях и без платы за пользование патентом. Перед первым туром конкурса в NIST поступило 21 предложение, 15 из которых соответствовали выдвинутым критериям. Затем были проведены исследования этих решений, в том числе связанные с дешифровкой и проверкой производительности, и получены экспертные оценки специалистов по криптографии. В августе 1999 года NIST объявил пять финалистов, которые получили право на участие во втором этапе обсуждений. 2 октября 2000 года NIST сообщил о своем выборе - победителем конкурса стал алгоритм RIJNDAEL (произносится как «райндол») бельгийских криптографов Винсента Раймана и Йоана Дамана, который зарегистрирован в качестве официального федерального стандарта как FIPS 197 (Federal Information Processing Standard).

Так как я уже реализовывал AES шифрование/дешифрование, то хотел бы рассмотреть алгоритм блочного шифрования Camellia, который так же является одним из финалистов европейского конкурса NESSIE (наряду с AES и Shacal-2).

3. CAMELLIA - ГЕНЕРАЦИЯ ВСПОМОГАТЕЛЬНЫХ КЛЮЧЕЙ

1. Ключ (K) разбивается на 2 128-битные части KL и KR.

Ключ	KL	KR
128	K	0
192	K » 64	((K & MASK64) « 64) | ((K & MASK64))
256	K » 128	K & MASK128

2. Вычисляем 128-битные числа KA и KB (см. схему). Переменные D1 и D2 64-битные.

```
1     D1 = (KL^ KR) >> 64;
2     D2 = (KL ^ KR) & MASK64;
3     D2 = D2 ^ F(D1, C1);
4     D1 = D1 ^ F(D2, C2);
5     D1 = D1 ^ (KL >> 64);
6     D2 = D2 ^ (KL & MASK64);
7     D2 = D2 ^ F(D1, C3);
8     D1 = D1 ^ F(D2, C4);
9     KA = (D1 << 64) | D2;
10    D1 = (KA ^ KR) >> 64;
11    D2 = (KA ^ KR) & MASK64;
12    D2 = D2 ^ F(D1, C5);
13    D1 = D1 ^ F(D2, C6);
14    KB = (D1 << 64) | D2;
15
```

3. Вычисляем вспомогательные 64-битные ключи kw1, ..., kw4, k1, ..., k24, ke1, ..., ke6 в зависимости от размера ключа:

Для ключа 128 бит:

```
1     kw1 = (KL <<< 0) >> 64;
2
```

```
1     kw2 = (KL <<<   0) & MASK64;
2     k1  = (KA <<<   0) >> 64;
3     k2  = (KA <<<   0) & MASK64;
4     k3  = (KL <<<  15) >> 64;
5     k4  = (KL <<<  15) & MASK64;
6     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;

```

Для ключей 192/256 бит:

```
    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;

```

4. CAMELLIA - ШИФРОВАНИЕ

Шифрование происходит по схеме Фейстеля с 18 этапами для 128-битного ключа и 24 этапами для 192- и 256-битных ключей. Каждые 6 этапов применяются функции FL и FLINV.

Для ключа 128 бит:

```
    D1 = M >> 64;
    // Cypher text devides in 2 64-bits parts

```

```
    D2 = M & MASK64;
    D1 = D1 ^ kw1;              //  prewhitening
    D2 = D2 ^ kw2;
    D2 = D2 ^ F(D1, k1);
    D1 = D1 ^ F(D2, k2);
    D2 = D2 ^ F(D1, k3);
    D1 = D1 ^ F(D2, k4);
    D2 = D2 ^ F(D1, k5);
    D1 = D1 ^ F(D2, k6);
    D1 = FL   (D1, ke1);        // FL
    D2 = FLINV(D2, ke2);        // FLINV
    D2 = D2 ^ F(D1, k7);
    D1 = D1 ^ F(D2, k8);
    D2 = D2 ^ F(D1, k9);
    D1 = D1 ^ F(D2, k10);
    D2 = D2 ^ F(D1, k11);
    D1 = D1 ^ F(D2, k12);
    D1 = FL   (D1, ke3);        // FL
    D2 = FLINV(D2, ke4);        // FLINV
    D2 = D2 ^ F(D1, k13);
    D1 = D1 ^ F(D2, k14);
    D2 = D2 ^ F(D1, k15);
    D1 = D1 ^ F(D2, k16);
    D2 = D2 ^ F(D1, k17);
    D1 = D1 ^ F(D2, k18);
    D2 = D2 ^ kw3;              // final whitening
    D1 = D1 ^ kw4;
    C = (D2 << 64) | D1;

```

Для ключей 192/256 бит:

```
    D1 = M >> 64;
    // Cypher text devides in 2 64-bits parts

```

```
    D2 = M & MASK64;
    D1 = D1 ^ kw1;              // prewhitening
    D2 = D2 ^ kw2;
    D2 = D2 ^ F(D1, k1);
    D1 = D1 ^ F(D2, k2);
    D2 = D2 ^ F(D1, k3);
    D1 = D1 ^ F(D2, k4);
    D2 = D2 ^ F(D1, k5);
    D1 = D1 ^ F(D2, k6);
    D1 = FL   (D1, ke1);        // FL
    D2 = FLINV(D2, ke2);        // FLINV
    D2 = D2 ^ F(D1, k7);
    D1 = D1 ^ F(D2, k8);
    D2 = D2 ^ F(D1, k9);
    D1 = D1 ^ F(D2, k10);
    D2 = D2 ^ F(D1, k11);
    D1 = D1 ^ F(D2, k12);
    D1 = FL   (D1, ke3);        // FL
    D2 = FLINV(D2, ke4);        // FLINV
    D2 = D2 ^ F(D1, k13);
    D1 = D1 ^ F(D2, k14);
    D2 = D2 ^ F(D1, k15);
    D1 = D1 ^ F(D2, k16);
    D2 = D2 ^ F(D1, k17);
    D1 = D1 ^ F(D2, k18);
    D1 = FL   (D1, ke5);        // FL
    D2 = FLINV(D2, ke6);        // FLINV
    D2 = D2 ^ F(D1, k19);
    D1 = D1 ^ F(D2, k20);
    D2 = D2 ^ F(D1, k21);
    D1 = D1 ^ F(D2, k22);
    D2 = D2 ^ F(D1, k23);
    D1 = D1 ^ F(D2, k24);
    D2 = D2 ^ kw3;              // final whitening
```

```
    D1 = D1 ^ kw4;
    C = (D2 << 64) | D1;

```

4.1. Вспомогательные функции F, FL, FLINV

$F-$, $FL-$ и $FLINV-$функции на вход получают 2 64-битных параметра - данные F_IN и ключ KE. Функция F использует 16 8-битных переменных $t1, ..., t8, y1, ..., y8$ и 1 64-битную переменную. На выходе функции 64-битное число. Функции FL и $FLINV$ используют 4 32-битные переменные $x1, x2, k1, k2$. На выходе функции 64-битное число. Функция $FLINV$ - обратная к FL.

F-функция

```
    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;

```

FL-функция

```
    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;

```

FLINV-функция

```
    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;

```

S - блоки

Значение функции $SBOX1$ определяется из таблицы S-BOX. Для примера: $SBOX1(0x7a) = 232$. $SBOX2$, $SBOX3$ и $SBOX4$ определяются из $SBOX1$ следующим образом:

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

```
3       SBOX4[x] = SBOX1[x <<< 1];
4
```

5. CAMELLIA - РАСШИФРОВАНИЕ

Алгоритм расшифрования идентичен шифрованию, с тем лишь различием, что вспомогательные ключи меняются местами по следующей схеме, в зависимости от длины исходного ключа:

Размер ключа

128 бит	192 или 256 бит
kw1 <-> kw3	kw1 <-> kw3
kw2 <-> kw4	kw2 <-> kw4
k1 <-> k18	k1 <-> k24
k2 <-> k17	k2 <-> k23
k3 <-> k16	k3 <-> k22
k4 <-> k15	k4 <-> k21
k5 <-> k14	k5 <-> k20
k6 <-> k13	k6 <-> k19
k7 <-> k12	k7 <-> k18
k8 <-> k11	k8 <-> k17
k9 <-> k10	k9 <-> k16
	k10 <-> k15
	k11 <-> k14
	k12 <-> k13
ke1 <-> ke4	ke1 <-> ke6
ke2 <-> ke3	ke2 <-> ke5
	ke3 <-> ke4

Пример шифрования

Ключ: 0123456789abcdeffedcba9876543210
Шифруемое сообщение: 0123456789abcdeffedcba9876543210
Зашифрованное сообщение: 67673138549669730857065648eabe43

6. CAMELLIA - РЕАЛИЗАЦИЯ НА C++

В данном разделе я продемонстрирую функции, которые отвечают за каждый конкретный шаг внутри программы без дополнительных пояснений, так как всё было рассказано ранее и алгоритм шифрования довольно прост в понимании и сопоставлении с простейшими битовыми операциями внутри программы.

```
1    void FeistelFunc(const uint8_t *x, const uint8_t *k
   , uint8_t *y) {
2      uint8_t t[8];
3      t[0] = SBOX1(x[0] ^ k[0]);
4      t[1] = SBOX2(x[1] ^ k[1]);
5      t[2] = SBOX3(x[2] ^ k[2]);
6      t[3] = SBOX4(x[3] ^ k[3]);
7      t[4] = SBOX2(x[4] ^ k[4]);
8      t[5] = SBOX3(x[5] ^ k[5]);
9      t[6] = SBOX4(x[6] ^ k[6]);
10     t[7] = SBOX1(x[7] ^ k[7]);
11
12     y[0] ^= t[0] ^ t[2] ^ t[3] ^ t[5] ^ t[6] ^ t[7];
13     y[1] ^= t[0] ^ t[1] ^ t[3] ^ t[4] ^ t[6] ^ t[7];
14     y[2] ^= t[0] ^ t[1] ^ t[2] ^ t[4] ^ t[5] ^ t[7];
15     y[3] ^= t[1] ^ t[2] ^ t[3] ^ t[4] ^ t[5] ^ t[6];
16     y[4] ^= t[0] ^ t[1] ^ t[5] ^ t[6] ^ t[7];
17     y[5] ^= t[1] ^ t[2] ^ t[4] ^ t[6] ^ t[7];
18     y[6] ^= t[2] ^ t[3] ^ t[4] ^ t[5] ^ t[7];
19     y[7] ^= t[0] ^ t[3] ^ t[4] ^ t[5] ^ t[6];
20   }
21
```

```
1    void FllayerFunc(uint8_t *x, const uint8_t *kl,
   const uint8_t *kr) {
2      uint32_t t[4], u[4], v[4];
3      B2W(x, t);
4      B2W(kl, u);
5      B2W(kr, v);
6
7      t[1] ^= (t[0] & u[0]) << 1 ^ (t[0] & u[0]) >> 31;
8      t[0] ^= t[1] | u[1];
```

```
    t[2] ^= t[3] | v[1];
    t[3] ^= (t[2] & v[0]) << 1 ^ (t[2] & v[0]) >> 31;

    W2BFunc(t, x);
  }

```

```
  void W2BFunc(const uint32_t *x, uint8_t *y) {
    for (int i = 0; i < 4; i++) {
      y[(i << 2) + 0] = (uint8_t)(x[i] >> 24 & 0xff);
      y[(i << 2) + 1] = (uint8_t)(x[i] >> 16 & 0xff);
      y[(i << 2) + 2] = (uint8_t)(x[i] >> 8 & 0xff);
      y[(i << 2) + 3] = (uint8_t)(x[i] >> 0 & 0xff);
    }
  }

```

```
  void ROTFunc(const uint32_t *x, const uint32_t n,
uint32_t *y) {
    int r = n & 31;
    if (r) {
      y[0] = x[((n >> 5) + 0) & 3] << r ^ x[((n >> 5)
 + 1) & 3] >> (32 - r);
      y[1] = x[((n >> 5) + 1) & 3] << r ^ x[((n >> 5)
 + 2) & 3] >> (32 - r);
    }
    else {
      y[0] = x[((n >> 5) + 0) & 3];
      y[1] = x[((n >> 5) + 1) & 3];
    }
  }

```

```
  void swapFunc(uint8_t *x) {
    for (int i = 0; i < 8; i++) {
      uint8_t tmp = x[i];
      x[i] = x[i + 8];
      x[i + 8] = tmp;
    }
  }

```

7. ЗАКЛЮЧЕНИЕ

В заключение хотелось бы сказать, что блочный шифр Camellia показался мне интересным и легким в реализации, особенно с учётом прошлого семестра, где нам объяснялась алгебра, которая тут очень сильно понадобилась. Хотелось бы и дальше реализовывать и исследовать такого рода шифры.