LCSSL – модифицированная версия OpenSSL с поддержкой алгоритмов ГОСТ Р34.10-2012 и ГОСТ Р34.11-2012 Руководство Программиста

ООО "ЛИССИ-Софт"

18 сентября 2014 г.

Оглавление

1 Введение

LCSSL – это модифицированная версия OpenSSL 1.0.0e [15], представленная библиотеками liblccrypto и liblcssl, утилитой командной строки lcssl, а также включаемыми файлами. В данную версию добавлены фрагменты кода, обрабатывающие объектные идентификаторы, необходимые для поддержки алгоритмов ГОСТ Р34.11-2012 [4], ГОСТ Р34.10-2012 [6], а также сопутствующих алгоритмов и параметров, определенных руководящими документами ТК 26.

2 Общие сведения

`lc_core_2012` – это динамическая библиотека (плагин или ENGINE в терминологии OpenSSL), предназначенная для поддержки российских криптографических алгоритмов в приложениях OpenSSL.

ENGINE `lc_core_2012` все криптографические операции реализует средствами библиотеки lcc2012. По сравнению со своим предшественником `lc_core`, `lc_core_2012` дополнительно поддерживает алгоритмы ГОСТ Р34.11-2012 и ГОСТ Р34.10-2012, а также сопутствующие алгоритмы и параметры, определенные руководящими документами ТК 26.

`lc_p11_core_2012` – это комбинированный ENGINE, предназначенная для поддержки российских криптографических алгоритмов в приложениях OpenSSL с использованием объектов ключевой пары на токене с интерфейсом PKCS#11 [16]. Данный ENGINE реализует интерфейс STORE для доступа к объектам токена.

ENGINE `lc_p11_core_2012` все криптографические операции реализует средствами библиотеки lcc2012, за исключением операций с закрытом ключом, которые осуществляются через интефейсы библиотеки PKCS#11 токена. Настройка данного ENGINE на конкретную библиотеку PKCS#11 производится управляющими командами программно или в файле конфигурации OpenSSL.

С точки зрения программиста, ENGINE – это обычная динамическая библиотека, экспортирующая определенный набор функций.

Утилита командной строки lcssl по функциональности аналогична штатной утилите openssl. Дополнительная утилита store позволяет выполнять из командной строки некоторые операции с объектами токена.

2.1 Системные требования

LCSSL функционирует во всех операционных системах, где работает OpenSSL версии 1.0.0 и выше.

2.2 Особенности реализации

`lc_core_2012` и `lc_p11_core_2012` зависят только от базовой криптографической библиотеки lcc2012 и библиотеки liblccrypto.

Для поддержки использования протокола TLS с ключами и сертификатами ГОСТ Р34.10-2001 и ГОСТ Р34.10-2012 (256 и 512 бит) в библиотеке liblcssl применяется расширенная трактовка шифрсьюта "GOST2001-GOST89-GOST89 определенного для ГОСТ Р34.10-2001. GOST2001 рассматривается в LCSSL как семейство алго-

ритмов ГОСТ Р34.10-2001 и ГОСТ Р34.10-2012 (256 и 512 бит). Особенностью такой поддержки является использование базового алгоритма хэширования ГОСТ Р34.11-94 во всех внутренних функциях хэндшейка, не зависящих от типа используемых ключей, включая TLS PRF. После утверждения новых шифрсьютов в библиотеку liblcssl будут внесены изменения для их поддержки в соответствии с руководящими документами ТК 26.

Следует иметь в виду, что некоторые использованные в данном проекте руководящие документы ТК 26 еще не утверждены и только готовятся к публикации. Поэтому в дальнейших версиях проекта возможны соответствующие изменения.

3 Установка

3.1 Windows

Все ENGINE устанавливаются в подкаталог lib/engines в каталоге установки LCSSL. Библиотеки liblccrypto и liblcssl, а также утилиты командной строки lcssl и store устанавливаются в подкаталог bin. Включаемые файлы устанавливаются в подкаталог include/openssl. Умалчиваемый файл конфигурации openssl.cnf для lcssl размещается в подкаталоге ssl. Для удобства работы с утилитами полный путь к файлу конфигурации следует прописать в переменной среды `LCSSL_CONF`. Полный путь к папке lib/engines также следует прописать в переменной среды `LCSSL_ENGINES`. Полный путь к динамическим библиотекам и выполняемым файлам следует добавить к значению переменной среды PATH.

3.2 Linux

ENGINE `lc_core_2012` устанавливается в подкаталог /usr/local/lib/engines. Библиотеки liblccrypto и liblcssl устанавливаются в папку /usr/local/lib, а утилиты командной строки lcssl и store устанавливаются в /usr/local/bin. Умалчиваемый файл конфигурации openssl.cnf для lcssl размещается в папке /etc/lissi-soft/lcssl. Включаемые файлы устанавливаются в папку /usr/local/include/lcssl/openssl. Для удобства работы с утилитами полный путь к файлу конфигурации следует прописать в переменной среды `LCSSL_CONF`. Полный путь к папке /usr/local/lib/engines также следует прописать в переменной среды `LCSSL_ENGINES`. Полный путь к динамическим библиотекам /usr/local/lib следует добавить к значению переменной среды `LD_LIBRARY_PATH`.

4 Лицензирование

Утилиты командной строки lcssl, store и библиотеки liblccrypto, liblcssl распространяются свободно и не требуют лицензирования. Однако, перед использованием `lc_core_2012` и `lc_p11_core_2012` должны быть лицензированы на сайте ООО "ЛИССИ-Софт"[1].

5 Примеры программ

5.1 Генерация дайджеста

Листинг 5.1: lc_core_2012_digest.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/engine.h>
#include <openssl/err.h>
#include "getopt.h"

void bio_print_hex(BIO *bio, unsigned char *buf, size_t len) {
    size_t i;
    for (i=0; i<len; i++)
        BIO_printf(bio, "%02X%c", buf[i],
        ((i+1)==len||!((i+1)%16))?'\n':' ');
}

BIO *bio_err = NULL;
BIO *bio_out = NULL;
ENGINE *e = NULL;
// Пример для ГОСТ Р 34.11-94
unsigned char data_94[] = {
  0x61, 0x62, 0x63, 0x64, 0x62, 0x63, 0x64, 0x65,
  0x63, 0x64, 0x65, 0x66, 0x64, 0x65, 0x66, 0x67,
  0x65, 0x66, 0x67, 0x68, 0x66, 0x67, 0x68, 0x69,
  0x67, 0x68, 0x69, 0x6A, 0x68, 0x69, 0x6A, 0x6B,
  0x69, 0x6A, 0x6B, 0x6C, 0x6A, 0x6B, 0x6C, 0x6D,
  0x6B, 0x6C, 0x6D, 0x6E, 0x6C, 0x6D, 0x6E, 0x6F,
  0x6D, 0x6E, 0x6F, 0x70, 0x6E, 0x6F, 0x70, 0x71,
  0x0A
};
unsigned char et_94[] = {
  0xc8, 0x77, 0xc2, 0xd1, 0x7f, 0xd3, 0x99, 0x2e,
```

```
  0x7a, 0x97, 0xc5, 0x67, 0x07, 0xdf, 0x57, 0xc0,
  0x5b, 0xdc, 0xf2, 0x17, 0x34, 0x6a, 0x69, 0x2f,
  0x6b, 0x9a, 0xad, 0xc4, 0x47, 0xa8, 0x2f, 0xd2
};
// Примеры из ГОСТ Р 34.11-2012:
unsigned char M1[] = {
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    0x38, 0x39, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35,
    0x36, 0x37, 0x38, 0x39, 0x30, 0x31, 0x32, 0x33,
    0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30, 0x31,
    0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    0x38, 0x39, 0x30, 0x31, 0x32, 0x33, 0x34, 0x35,
    0x36, 0x37, 0x38, 0x39, 0x30, 0x31, 0x32,
};
unsigned char et1_32[] = {
    0x9d, 0x15, 0x1e, 0xef, 0xd8, 0x59, 0x0b, 0x89,
    0xda, 0xa6, 0xba, 0x6c, 0xb7, 0x4a, 0xf9, 0x27,
    0x5d, 0xd0, 0x51, 0x02, 0x6b, 0xb1, 0x49, 0xa4,
    0x52, 0xfd, 0x84, 0xe5, 0xe5, 0x7b, 0x55, 0x00,
};
unsigned char et1_64[] = {
    0x1b, 0x54, 0xd0, 0x1a, 0x4a, 0xf5, 0xb9, 0xd5,
    0xcc, 0x3d, 0x86, 0xd6, 0x8d, 0x28, 0x54, 0x62,
    0xb1, 0x9a, 0xbc, 0x24, 0x75, 0x22, 0x2f, 0x35,
    0xc0, 0x85, 0x12, 0x2b, 0xe4, 0xba, 0x1f, 0xfa,
    0x00, 0xad, 0x30, 0xf8, 0x76, 0x7b, 0x3a, 0x82,
    0x38, 0x4c, 0x65, 0x74, 0xf0, 0x24, 0xc3, 0x11,
    0xe2, 0xa4, 0x81, 0x33, 0x2b, 0x08, 0xef, 0x7f,
    0x41, 0x79, 0x78, 0x91, 0xc1, 0x64, 0x6f, 0x48,
};
unsigned char M2[] = {
    0xd1, 0xe5, 0x20, 0xe2, 0xe5, 0xf2, 0xf0, 0xe8,
    0x2c, 0x20, 0xd1, 0xf2, 0xf0, 0xe8, 0xe1, 0xee,
    0xe6, 0xe8, 0x20, 0xe2, 0xed, 0xf3, 0xf6, 0xe8,
    0x2c, 0x20, 0xe2, 0xe5, 0xfe, 0xf2, 0xfa, 0x20,
    0xf1, 0x20, 0xec, 0xee, 0xf0, 0xff, 0x20, 0xf1,
    0xf2, 0xf0, 0xe5, 0xeb, 0xe0, 0xec, 0xe8, 0x20,
    0xed, 0xe0, 0x20, 0xf5, 0xf0, 0xe0, 0xe1, 0xf0,
    0xfb, 0xff, 0x20, 0xef, 0xeb, 0xfa, 0xea, 0xfb,
    0x20, 0xc8, 0xe3, 0xee, 0xf0, 0xe5, 0xe2, 0xfb,
};
unsigned char et2_32[] = {
    0x9d, 0xd2, 0xfe, 0x4e, 0x90, 0x40, 0x9e, 0x5d,
```

```
    0xa8, 0x7f, 0x53, 0x97, 0x6d, 0x74, 0x05, 0xb0,
    0xc0, 0xca, 0xc6, 0x28, 0xfc, 0x66, 0x9a, 0x74,
    0x1d, 0x50, 0x06, 0x3c, 0x55, 0x7e, 0x8f, 0x50,
};
unsigned char et2_64[] = {
    0x1e, 0x88, 0xe6, 0x22, 0x26, 0xbf, 0xca, 0x6f,
    0x99, 0x94, 0xf1, 0xf2, 0xd5, 0x15, 0x69, 0xe0,
    0xda, 0xf8, 0x47, 0x5a, 0x3b, 0x0f, 0xe6, 0x1a,
    0x53, 0x00, 0xee, 0xe4, 0x6d, 0x96, 0x13, 0x76,
    0x03, 0x5f, 0xe8, 0x35, 0x49, 0xad, 0xa2, 0xb8,
    0x62, 0x0f, 0xcd, 0x7c, 0x49, 0x6c, 0xe5, 0xb3,
    0x3f, 0x0c, 0xb9, 0xdd, 0xdc, 0x2b, 0x64, 0x60,
    0x14, 0x3b, 0x03, 0xda, 0xba, 0xc9, 0xfb, 0x28,
};

int do_crypt_94() {
    EVP_MD_CTX *md_ctx = NULL;
    unsigned char hash[32];
    size_t hash_len = 32;
  int rc;
  int i;
  EVP_MD *md = NULL;

    // Make message digest with e
  BIO_printf(bio_err, "Make GOST R34.11-94 data_94 digest\n");
//  md = ENGINE_get_digest(e, NID_id_GostR3411_94);
  md = EVP_get_digestbyname("md_gost94");
  if(!md) {
    rc = 0;
    goto err;
  }
  md_ctx = EVP_MD_CTX_create();
  rc = EVP_DigestInit_ex(md_ctx, md, e);
  if (rc != 1) {
    goto err;
  }
  rc = EVP_DigestUpdate(md_ctx, data_94, sizeof(data_94));
  if (rc != 1) {
    goto err;
  }
  rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
  if (rc != 1) {
    goto err;
  }
```

```
  EVP_MD_CTX_destroy(md_ctx);
  bio_print_hex(bio_err, hash, sizeof(hash));
  if (memcmp(hash, et_94, sizeof(et_94)) != 0) {
    BIO_printf(bio_err, "Invalid data_94 digest\n");
    goto err;
  }
  BIO_printf(bio_err, "data_94 digest OK\n");

  return 1;
err:
    BIO_printf(bio_err, "Error 0x%x\n", rc);
  return 0;
}

int do_crypt_256() {
    EVP_MD_CTX *md_ctx = NULL;
    unsigned char hash[32];
    size_t hash_len = 32;
  int rc;
  int i;
  EVP_MD *md = NULL;

    // Make message digest with e
  BIO_printf(bio_err, "Make GOST R34.11-2012-256 M1 digest\n");
//  md = ENGINE_get_digest(e, NID_id_tc26_gost3411_2012_256);
  // Можно запросить EVP_MD и по SN-имени:
  md = EVP_get_digestbyname("gost3411-2012-256");
  if(!md) {
    rc = 0;
    goto err;
  }
  md_ctx = EVP_MD_CTX_create();
  rc = EVP_DigestInit_ex(md_ctx, md, e);
  if (rc != 1) {
    goto err;
  }
  rc = EVP_DigestUpdate(md_ctx, M1, sizeof(M1));
  if (rc != 1) {
    goto err;
  }
  rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
  if (rc != 1) {
    goto err;
  }
```

```
  EVP_MD_CTX_destroy(md_ctx);
  bio_print_hex(bio_err, hash, sizeof(hash));
  if (memcmp(hash, et1_32, sizeof(et1_32)) != 0) {
    BIO_printf(bio_err, "Invalid M1 digest\n");
    goto err;
  }
  BIO_printf(bio_err, "M1 digest OK\n");
  BIO_printf(bio_err,
    "-------------------------------------------------------\n");

  BIO_printf(bio_err, "Make GOST R34.11-2012-256 M2 digest\n");
  md_ctx = EVP_MD_CTX_create();
  rc = EVP_DigestInit_ex(md_ctx, md, e);
  if (rc != 1) {
    goto err;
  }
  rc = EVP_DigestUpdate(md_ctx, M2, sizeof(M2));
  if (rc != 1) {
    goto err;
  }
  rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
  if (rc != 1) {
    goto err;
  }
  EVP_MD_CTX_destroy(md_ctx);
  bio_print_hex(bio_err, hash, sizeof(hash));
  if (memcmp(hash, et2_32, sizeof(et2_32)) != 0) {
    BIO_printf(bio_err, "Invalid M2 digest\n");
    goto err;
  }
  BIO_printf(bio_err, "M2 digest OK\n");
  return 1;
err:
    BIO_printf(bio_err, "Error 0x%x\n", rc);
  return 0;
}

int do_crypt_512() {
    EVP_MD_CTX *md_ctx = NULL;
    unsigned char hash[64];
    size_t hash_len = 64;
  int rc;
  int i;
  EVP_MD *md = NULL;
```

```
  // Make message digest with e
BIO_printf(bio_err, "Make GOST R34.11-2012-512 M1 digest\n");
md = ENGINE_get_digest(e, NID_id_tc26_gost3411_2012_512);
if(!md) {
  rc = 0;
  goto err;
}
md_ctx = EVP_MD_CTX_create();
rc = EVP_DigestInit_ex(md_ctx, md, e);
if (rc != 1) {
  goto err;
}
rc = EVP_DigestUpdate(md_ctx, M1, sizeof(M1));
if (rc != 1) {
  goto err;
}
rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
if (rc != 1) {
  goto err;
}
EVP_MD_CTX_destroy(md_ctx);
bio_print_hex(bio_err, hash, sizeof(hash));
if (memcmp(hash, et1_64, sizeof(et1_64)) != 0) {
  BIO_printf(bio_err, "Invalid M1 digest\n");
  goto err;
}
BIO_printf(bio_err, "M1 digest OK\n");
BIO_printf(bio_err,
  "-------------------------------------------------------\n");

BIO_printf(bio_err, "Make GOST R34.11-2012-512 M2 digest\n");
md_ctx = EVP_MD_CTX_create();
rc = EVP_DigestInit_ex(md_ctx, md, e);
if (rc != 1) {
  goto err;
}
rc = EVP_DigestUpdate(md_ctx, M2, sizeof(M2));
if (rc != 1) {
  goto err;
}
rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
if (rc != 1) {
  goto err;
```

```
  }
  EVP_MD_CTX_destroy(md_ctx);
  bio_print_hex(bio_err, hash, sizeof(hash));
  if (memcmp(hash, et2_64, sizeof(et2_64)) != 0) {
    BIO_printf(bio_err, "Invalid M2 digest\n");
    goto err;
  }
  BIO_printf(bio_err, "M2 digest OK\n");
  return 1;
err:
    BIO_printf(bio_err, "Error 0x%x\n", rc);
  return 0;
}

int main(int argc, char **argv)
{
  unsigned int j;
    int rc;
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
    extern char *optarg;
    int c;

  while((c = getopt(argc, argv, "e:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
```

```
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    // Builtin engines should be loaded first
    // to access builtin "dynamic" engine then.
    ENGINE_load_builtin_engines();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load lprng_engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
            BIO_printf(bio_err, "Loading %s ENGINE failed\n",
   engine_path);
            goto err;
    }
  }

  BIO_printf(bio_err,
    "Testing GOST R34.11-94, GOST R34.11-2012-256 and \n"
    "GOST R34.11-2012-512 digests with %s engine\n"
    "=======================================================\n",
    engine_path);

  rc = do_crypt_94();
  if (rc != 1) {
    goto err;
  }
  BIO_printf(bio_err,
    "=======================================================\n");

  rc = do_crypt_256();
  if (rc != 1) {
    goto err;
  }
  BIO_printf(bio_err,
    "=======================================================\n");
```

```
  rc = do_crypt_512();
  if (rc != 1) {
    goto err;
  }
  BIO_printf(bio_err,
    "============================================================\n");

  ENGINE_free(e);
  e = NULL;

    BIO_printf(bio_out, "SUCCESS\n");
  if (engine_path)
    OPENSSL_free(engine_path);
  if (bio_out)
    BIO_free(bio_out);

    rc = 0;
    return rc;

err:
    if (e)
        ENGINE_free(e);
  if (engine_path)
    OPENSSL_free(engine_path);
  ERR_print_errors(bio_err);
  if (bio_out)
    BIO_free(bio_out);

  rc = -1;
    return rc;
}
```

5.2 Шифрование

Установка умалчиваемого набора параметров шифрования для ГОСТ 28147-89 обычно производится управляющей командой при загрузке ENGINE:

```
ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
"id-Gost28147-89-CryptoPro-A-ParamSet", 0)
```

Установить другой набор параметров в контексте шифрования можно динамически другой управляющей командой сразу после инициализации контекста, например:

```
EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GOST28147_CIPHER_PARAMSET,
NID_id_tc26_gost_28147_89_param_Z, NULL)
```

Листинг 5.2: lc_core_2012_evp_cipher.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include <sys/types.h>
#include <sys/timeb.h>
#ifndef WIN32
#include <sys/time.h>
#endif

#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/engine.h>
#include <openssl/rand.h>
#include <openssl/err.h>
#include "getopt.h"
#include "lc_defs_2012.h"

void bio_print_hex(BIO *bio, unsigned char *buf, size_t len) {
    size_t i;
    for (i=0; i<len; i++)
        BIO_printf(bio, "%02X%c", buf[i],
        ((i+1)==len||!((i+1)%16))?'\n':' ');
}

BIO *bio_err = NULL;
BIO *bio_out = NULL;

#define SYSTEMTIME  struct timeb
#define GetSystemTime(x)    ftime(x)

long process_time(struct timeb t1, struct timeb t2)
{
  long ms    = t2.millitm - t1.millitm;
  long s     = t2.time - t1.time;

  while (ms < 0) {
    ms += 1000;
    s--;
  }
  ms += (s*1000);
```

```
  printf("Time:  %ld msec\n", ms );
  return ms;
}

int main(int argc,char **argv)
{
  ENGINE *e = NULL;
    const EVP_CIPHER *cipher;
    EVP_CIPHER_CTX *ctx;
    unsigned char key[32];
    unsigned char iv[8];
    char plain_text[] = "Plain text for encrypting and
   decrypting";
    int plain_text_len = strlen(plain_text);
    char cipher_text[128];
    int cipher_text_len = sizeof(cipher_text);
    char decrypted_text[128];
    int decrypted_text_len = 0;
    int dummy_len = 0;
    int rc = -1;
  char *lc_core_path = "lc_core_2012";
  int i;
  SYSTEMTIME t1, t2;
  long diff;
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
    extern char *optarg;
    int c;

  while((c = getopt(argc, argv, "e:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
      }
```

```
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  BIO_printf(bio_out, "%s EVP_CIPHER test\n", engine_path);

    // Builtin engines should be loaded first
    // to access builtin "dynamic" engine then.
    ENGINE_load_builtin_engines();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load lprng_engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0)
      || !ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
        "id-Gost28147-89-CryptoPro-A-ParamSet", 0))
//        "id-tc26-gost-28147-89-param-A", 0))
    {
          BIO_printf(bio_err, "Loading %s ENGINE failed\n",
   engine_path);
          goto err;
    }
  }

#ifdef LISSI_DEBUG
  BIO_printf(bio_err, "Engine 0x%x loaded with id '%s'\n",
    e, ENGINE_get_id(e));
#endif

    BIO_printf(bio_out, "Get EVP_CIPHER by NID\n");
    cipher = ENGINE_get_cipher(e, NID_id_Gost28147_89);
    if(!cipher) {
```

```
        BIO_printf(bio_err,
      "Can't get cipher NID_id_Gost28147_89\n");
        rc = -1;
      goto err;
    }

    BIO_printf(bio_out,
    "Generate random cipher key\n");
  if (RAND_bytes(key, sizeof(key)) < 0) {
        BIO_printf(bio_err, "RAND_bytes failed\n");
    rc = -1;
    goto err;
  }
    BIO_printf(bio_out, "Random cipher key:\n");
    bio_print_hex(bio_out, key, sizeof(key));

    BIO_printf(bio_out, "Generate random IV\n");
  if (RAND_bytes(iv, sizeof(iv)) < 0) {
        BIO_printf(bio_err, "RAND_bytes failed\n");
    rc = -1;
    goto err;
  }
    BIO_printf(bio_out, "Random IV:\n");
    bio_print_hex(bio_out, iv, sizeof(iv));

//GetSystemTime(&t1);
//for (i=0; i< 100000; i++)
//{

//    BIO_printf(bio_out, "Create new EVP_CIPHER_CTX\n");
    ctx = EVP_CIPHER_CTX_new();
    if (!ctx) {
        BIO_printf(bio_err, "EVP_CIPHER_CTX_new failed\n");
        rc = -1;
        goto err;
    }
//    BIO_printf(bio_out, "EVP_EncryptInit_ex\n");
    rc = EVP_EncryptInit_ex(ctx, cipher, e, key, iv);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_EncryptInit_ex failed\n");
        rc = -1;
        goto err;
    }
```

```
  rc = EVP_CIPHER_CTX_ctrl(ctx,
   EVP_CTRL_GOST28147_CIPHER_PARAMSET,
    NID_id_tc26_gost_28147_89_param_A, NULL);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_CIPHER_CTX_ctrl failed\n");
        rc = -1;
        goto err;
    }

//    BIO_printf(bio_out, "EVP_EncryptUpdate\n");
    rc = EVP_EncryptUpdate(ctx, cipher_text, &cipher_text_len,
        plain_text, plain_text_len);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_EncryptUpdate failed\n");
        rc = -1;
        goto err;
    }

    // No padding so use dummy text buffer here
  // for extra control
//    BIO_printf(bio_out, "EVP_EncryptFinal_ex\n");
    rc = EVP_EncryptFinal_ex(ctx, NULL, &dummy_len);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_EncryptFinal_ex failed\n");
        rc = -1;
        goto err;
    }
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_EncryptFinal failed\n");
        rc = -1;
        goto err;
    }

    if (dummy_len != 0) {
        BIO_printf(bio_err,
      "Non-empty padding in EVP_EncryptFinal_ex\n");
        rc = -1;
        goto err;
    }

//    BIO_printf(bio_out, "Plain text: '%s'\n", plain_text);
    BIO_printf(bio_out, "Cipher text:\n");
    bio_print_hex(bio_out, cipher_text, cipher_text_len);
    EVP_CIPHER_CTX_cleanup(ctx);
```

```
    memset(decrypted_text, 0, sizeof(decrypted_text));
//    BIO_printf(bio_out, "EVP_DecryptInit_ex\n");
    rc = EVP_DecryptInit_ex(ctx, cipher, e, key, iv);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_DecryptInit_ex failed\n");
        rc = -1;
        goto err;
    }

  rc = EVP_CIPHER_CTX_ctrl(ctx,
   EVP_CTRL_GOST28147_CIPHER_PARAMSET,
    NID_id_tc26_gost_28147_89_param_A, NULL);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_CIPHER_CTX_ctrl failed\n");
        rc = -1;
        goto err;
    }

//    BIO_printf(bio_out, "EVP_DecryptUpdate\n");
    rc = EVP_DecryptUpdate(ctx,
    decrypted_text, &decrypted_text_len,
        cipher_text, cipher_text_len);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_DecryptUpdate failed\n");
        rc = -1;
        goto err;
    }
//    BIO_printf(bio_out, "EVP_DecryptFinal_ex\n");
    // There is no padding for stream ciphers
  // so use dummy text buffer here for extra control.
    rc = EVP_DecryptFinal_ex(ctx, NULL, &dummy_len);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_DecryptFinal_ex failed\n");
        rc = -1;
        goto err;
    }
    if (dummy_len != 0) {
        BIO_printf(bio_err,
      "Non-empty padding in EVP_DecryptFinal_ex\n");
        rc = -1;
        goto err;
    }
    EVP_CIPHER_CTX_cleanup(ctx);
```

```
    EVP_CIPHER_CTX_free(ctx);

//}
//GetSystemTime(&t2);
//diff = process_time(t1, t2);

    BIO_printf(bio_out, "Check decrypted text\n");
    if (decrypted_text_len != plain_text_len) {
        BIO_printf(bio_err, "Invalid decrypted text length\n");
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Decrypted text: '%s'\n", decrypted_text
   );
    if (memcmp(decrypted_text, plain_text, decrypted_text_len)
   != 0) {
        BIO_printf(bio_err, "Invalid decrypted text\n");
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "SUCCESS\n");

  if (engine_path)
    OPENSSL_free(engine_path);
    ENGINE_cleanup();
    EVP_cleanup();
    CRYPTO_cleanup_all_ex_data();
    ERR_remove_thread_state(NULL);
    ERR_free_strings();
    CRYPTO_mem_leaks_fp(stderr);
    rc = 0;
  return rc;
err:
  if (engine_path)
    OPENSSL_free(engine_path);
    ENGINE_cleanup();
    EVP_cleanup();
    CRYPTO_cleanup_all_ex_data();
    ERR_remove_thread_state(NULL);
    ERR_free_strings();
    CRYPTO_mem_leaks_fp(stderr);
  rc = -1;
    return rc;
}
```

5.3 Имитовставка

Листинг 5.3: lc_core_2012_mac.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/bio.h>
#include <openssl/evp.h>
#include <openssl/crypto.h>
#include <openssl/engine.h>
#include <openssl/err.h>
#include "lc_defs_2012.h"

void bio_print_hex(BIO *bio, unsigned char *buf, size_t len) {
    size_t i;
    for (i=0; i<len; i++)
        BIO_printf(bio, "%02X%c", buf[i],
        ((i+1)==len||!((i+1)%16))?'\n':' ');
}

BIO *bio_err = NULL;
BIO *bio_out = NULL;
ENGINE *e = NULL;

int do_crypt() {
  EVP_MD *md = NULL;
    EVP_MD_CTX *md_ctx = NULL;
  EVP_PKEY_CTX *pctx = NULL;
  EVP_PKEY *pkey = NULL;
    unsigned char mac[4];
    size_t mac_len = 4;
  unsigned char key[] = {
      0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
      0x38, 0x39, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66,
      0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
      0x38, 0x39, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66
    };
    unsigned char iv[] = {
      0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37
    };
```

```
unsigned char data[] = {
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    0x38, 0x39, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    0x38, 0x39, 0x61, 0x62, 0x63, 0x64, 0x65, 0x66,

};
unsigned char et_A[] = {
  0x21, 0x78, 0x1e, 0xcb,
};
// При заданном IV:
unsigned char et2_A[] = {
  0xb7, 0x6e, 0xb5, 0x15,
};
unsigned char et_Z[] = {
  0x17, 0x0e, 0xfc, 0x07,
};
// При заданном IV:
unsigned char et2_Z[] = {
  0x30, 0x4f, 0x45, 0x86,
};
int rc;

  // Make message digest with e
BIO_printf(bio_err, "Make GOST 28147-89 MAC\n");
md = ENGINE_get_digest(e, NID_id_Gost28147_89_MAC);
if(!md) {
  goto err;
}

// Создаем контекст имитовставки
md_ctx = EVP_MD_CTX_create();
pkey = EVP_PKEY_new_mac_key(NID_id_Gost28147_89_MAC,
  e, key, sizeof(key));
if (!pkey) {
  goto err;
}
rc = EVP_DigestSignInit(md_ctx, &pctx, md, e, pkey);
if (rc != 1) {
  goto err;
}

// На самом деле имитовставка - это не дайджест,
// а MAC, причем у алгоритма ГОСТ 28147-89 входные
```

```
  // данные не могут быть пустыми, поэтому
  // EVP_DigestUpdate обязателен, причем суммарный
  // размер входных данных не может быть нулевым.
  rc = EVP_DigestUpdate(md_ctx, data, sizeof(data));
  if (rc != 1) {
    goto err;
  }

  rc = EVP_DigestSignFinal(md_ctx, mac, &mac_len);
  if (rc != 1) {
    goto err;
  }
  EVP_MD_CTX_destroy(md_ctx);
  bio_print_hex(bio_err, mac, mac_len);
  if (memcmp(mac, et_A, sizeof(et_A)) != 0) {
    BIO_printf(bio_err, "Invalid data MAC A\n");
    goto err;
  }
  BIO_printf(bio_err, "data MAC A OK\n");

  // Создаем контекст имитовставки
  md_ctx = EVP_MD_CTX_create();
  pkey = EVP_PKEY_new_mac_key(NID_id_Gost28147_89_MAC,
    e, key, sizeof(key));
  if (!pkey) {
    goto err;
  }
  rc = EVP_DigestSignInit(md_ctx, &pctx, md, e, pkey);
  if (rc != 1) {
    goto err;
  }
  // Установку параметра, отличного от умалчиваемого, нужно
  // делать управляющей командой EVP_MD_CTRL_MAC_PARAMSET
  // после инициализации контекста (чтобы в контексте уже был
  // установлен ключ шифрования).
  // При этом, в lc_core_2012 выполняется повторная внутренняя
  // инициализация контекста для заданного параметра.
  // Как ни странно, функция EVP_MD_ctrl в OpenSSL
  // не предусмотрена, хотя в структуре EVP_MD в evp.h
  // соответствующий указатель на функцию md_ctrl имеется.
  rc = md->md_ctrl(md_ctx, EVP_MD_CTRL_MAC_PARAMSET,
    NID_id_tc26_gost_28147_89_param_A, NULL);
  if (rc != 1) {
    goto err;
```

```
  }
  // Если ключ шифрования шифруется с помощью KEK,
  // то при вычислении имитовставки нужно задавать IV.
  // Это можно сделать управляющей командой
   EVP_MD_CTRL_MAC_SET_IV.
  // Заметим, что нельзя просто добавить IV в качестве
  // первого блока данных функцией EVP_DigestUpdate,
  // так как начальный блок при инициализации с помощью
  // EVP_DigestSignInit уже установлен нулевым.
  rc = md->md_ctrl(md_ctx, EVP_MD_CTRL_MAC_SET_IV,
    -1, iv);
  if (rc != 1) {
    goto err;
  }

  rc = EVP_DigestUpdate(md_ctx, data, sizeof(data));
  if (rc != 1) {
    goto err;
  }
  rc = EVP_DigestSignFinal(md_ctx, mac, &mac_len);
  if (rc != 1) {
    goto err;
  }
  EVP_MD_CTX_destroy(md_ctx);
  bio_print_hex(bio_err, mac, mac_len);
  if (memcmp(mac, et2_Z, sizeof(et2_Z)) != 0) {
    BIO_printf(bio_err, "Invalid data MAC Z with IV\n");
    goto err;
  }
  BIO_printf(bio_err, "data MAC Z with IV OK\n");

  return 1;
err:
    BIO_printf(bio_err, "Error 0x%x\n", rc);
  return 0;
}


int main(int argc, char **argv)
{
  unsigned int j;
    int rc;
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
```

```
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
    extern char *optarg;
    int c;

  while((c = getopt(argc, argv, "e:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    // Builtin engines should be loaded first
    // to access builtin "dynamic" engine then.
    ENGINE_load_builtin_engines();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load lprng_engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
            BIO_printf(bio_err, "Loading %s ENGINE failed\n",
   engine_path);
```

```
            goto err;
        }
    }

    BIO_printf(bio_err,
        "Testing GOST 28147-89 MAC with %s engine\n"
        "=============================================\n",
        engine_path);

    rc = do_crypt();
    if (rc != 1) {
        goto err;
    }
    BIO_printf(bio_err,
        "=============================================\n");


    if (engine_path)
        OPENSSL_free(engine_path);
    ENGINE_free(e);
    e = NULL;

        BIO_printf(bio_out, "SUCCESS\n");
    if (bio_out)
        BIO_free(bio_out);
        return 0;

err:
    if (engine_path)
        OPENSSL_free(engine_path);
        if (e)
            ENGINE_free(e);
    ERR_print_errors(bio_err);
    if (bio_out)
        BIO_free(bio_out);
        return -1;
}
```

5.4 ЭЦП

Листинг 5.4: lc_core_2012_sign_verify.c

```
#include <stdio.h>
```

```
#include <string.h>
#include <stdlib.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/engine.h>
#include <openssl/store.h>
#include <openssl/err.h>
#include "lc_keypair_2012.h"
#include "lc_defs_2012.h"
#include "getopt.h"

void bio_print_hex(BIO *bio, unsigned char *buf, size_t len) {
    size_t i;
    for (i=0; i<len; i++)
        BIO_printf(bio, "%02X%c", buf[i],
        ((i+1)==len||!((i+1)%16))?'\n':' ');
}

BIO *bio_err = NULL;
BIO *bio_out = NULL;

int main(int argc, char **argv)
{
  ENGINE *e = NULL;
  STORE *st = NULL;
  int rc = 0;
//  const char* param_28147 = "id-Gost28147-89-CryptoPro-A-
   ParamSet";
  const char* param_28147 = "id-tc26-gost-28147-89-param-A";
  EVP_PKEY *pkey = NULL;
    EVP_PKEY_CTX *ctx = NULL;
  unsigned char msg[] = {
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x00,
  };
    EVP_MD *md = NULL;
    EVP_MD_CTX *md_ctx = NULL;
    unsigned char hash[64];
  int hash_len = sizeof(hash);
    unsigned char sig[128];
    size_t sig_len = sizeof(sig);
  EVP_PKEY_CTX *pectx = NULL;
  LC_GOST2012_KEYPAIR *lcc_gost2012_key = NULL;
```

```
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
  char *slot = NULL;
    extern char *optarg;
    int c;
  char label_2001[] = "test_keypair_2001";
  char label_2012_256[] = "test_keypair_2012_256";
  char label_2012_512[] = "test_keypair_2012_512";


  while((c = getopt(argc, argv, "e:a:s:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
        case 's':
      slot = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

    // Error messages output
    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    // Standard output
    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
```

```
        BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  BIO_printf(bio_err, "%s sign/verify test\n", engine_path);
    // Builtin engines should be loaded first
    // to access builtin "dynamic" engine then.
    ENGINE_load_builtin_engines();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load other engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
          BIO_printf(bio_err, "Loading ENGINE %s failed\n",
   engine_path);
          goto err;
    }
    if (api_path) {
      // В данном примере демонстрируется формирование engine_id
      // с явным указанием слота, если при вызове программы он
      // задан с флагом -s.
      char engine_id[128];
      if (slot) {
        sprintf(engine_id, "%s:%s", api_path, slot);
      } else {
        sprintf(engine_id, "%s", api_path);
      }
      if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", engine_id, 0)
        ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
   CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
// Вариант с колбэком для ввода PIN: ..., Verify, Prompt,
   Callback,
//        ||!ENGINE_ctrl_cmd(e, "LOGIN", 1, "Enter PIN:",
   EVP_read_pw_string, 0))
// Этот вариант будет работать и в такой форме по умолчанию:
//        ||!ENGINE_ctrl_cmd(e, "LOGIN", 1, NULL, NULL, 0))
// или даже в такой (здесь OpenSSL не дает задавать NULL в треть
   ем аргументе,
// поэтому задается пустая строка, а Verify по умолчанию полагае
   тся равным 1):
```

```
//          ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "", 0))
        {
          BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
          goto err;
        }
        // Получаем из ENGINE интерфейс STORE.
        st = STORE_new_engine(e);
        if (!st) {
          BIO_printf(bio_err, "ENGINE has no STORE interface\n");
          rc = -1;
          goto err;
        }
      }
    }
    BIO_printf(bio_err, "Engine 0x%x loaded with id '%s'\n",
      e, ENGINE_get_id(e));


    rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
              param_28147, 0);
      if (!rc) {
          BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
     ;
          goto err;
      }
    // Make message digest 3411-94
      BIO_printf(bio_out, "Make NID_id_GostR3411_94 message digest
     \n");
      md = ENGINE_get_digest(e, NID_id_GostR3411_94);
      if(!md) {
          BIO_printf(bio_err, "Can't get digest
     NID_id_GostR3411_94\n");
          rc = -1;
          goto err;
      }
      md_ctx = EVP_MD_CTX_create();
      rc = EVP_DigestInit_ex(md_ctx, md, e);
      if (rc != 1) {
          rc = -1;
          goto err;
      }
      rc = EVP_DigestUpdate(md_ctx, msg, sizeof(msg));
      if (rc != 1) {
          rc = -1;
```

```
        goto err;
    }
    rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
    if (rc != 1) {
        rc = -1;
        goto err;
    }
    EVP_MD_CTX_destroy(md_ctx);

    // Sign
    memset(sig, 0, sizeof(sig));
// Генерируем ключевую пару
BIO_printf(bio_out,
    "Generate NID_id_GostR3410_2001 key pair with %s\n",
    SN_id_GostR3410_2001_CryptoPro_A_ParamSet);
if (st) {
    OPENSSL_ITEM attr_keypair[] = {
            {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
            {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
            {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
            {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
            {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_GostR3410_2001;
    int parnid = NID_id_GostR3410_2001_CryptoPro_A_ParamSet;
    char *label = label_2001;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey = STORE_generate_key(st, attr_keypair, param_keypair);
    if (!pkey)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
```

```
      "Keypair '%s' with algorithm %s and paramset %s generated
   in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  } else {
    pctx = EVP_PKEY_CTX_new_id(NID_id_GostR3410_2001, e);
    if (!pctx) {
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(pctx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(pctx, -1, -1,
   EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_GostR3410_2001_CryptoPro_A_ParamSet, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen(pctx, &pkey) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(pctx);
  }
  lcc_gost2012_key = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(pkey);
  if (lcc_gost2012_key->is_real_priv_key) {
    BIO_printf(bio_out, "Generated private key:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_value,
      lcc_gost2012_key->priv_key_len);
  } else {
    BIO_printf(bio_out, "Generated private key ID:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_id,
      sizeof(lcc_gost2012_key->priv_key_id));
  }
    // Create Sign context with ENGINE
    BIO_printf(bio_out, "Create sign context\n");
    ctx = EVP_PKEY_CTX_new(pkey, e);
    if (ctx) {
```

```
        // Sign hash data
        BIO_printf(bio_out, "Sign message hash\n");
        rc = EVP_PKEY_sign_init(ctx);
        if (rc == 1) {
            rc = EVP_PKEY_sign(ctx, sig, &sig_len, hash,
 hash_len);
            if (!rc) {
                rc = -1;
                goto err;
            }
            BIO_printf(bio_out, "Signature generated OK\n");
        } else {
            rc = -1;
            goto err;
        }
        EVP_PKEY_CTX_free(ctx);
    }

  // Verify
    // Create Verify context with ENGINE
    BIO_printf(bio_out, "Create verify context\n");
    ctx = EVP_PKEY_CTX_new(pkey, e);
    if (ctx) {
        // Verify signature
        BIO_printf(bio_out, "Verify message hash signature\n");
        rc = EVP_PKEY_verify_init(ctx);
        if (rc == 1) {
            rc = EVP_PKEY_verify(ctx, sig, sig_len, hash,
 hash_len);
            if (!rc) {
                rc = -1;
                goto err;
            }
            BIO_printf(bio_out, "Signature verified OK\n");
        } else {
            rc = -1;
            goto err;
        }
        EVP_PKEY_CTX_free(ctx);
    }
    EVP_PKEY_free(pkey);
  pkey = NULL;

  // Make message digest 3411-2012-256
```

```
    BIO_printf(bio_out,
    "Make NID_id_tc26_gost3411_2012_256 message digest\n");
    md = ENGINE_get_digest(e, NID_id_tc26_gost3411_2012_256);
    if(!md) {
        BIO_printf(bio_err,
      "Can't get digest NID_id_tc26_gost3411_2012_256\n");
        rc = -1;
        goto err;
    }
    md_ctx = EVP_MD_CTX_create();
    rc = EVP_DigestInit_ex(md_ctx, md, e);
    if (rc != 1) {
        rc = -1;
        goto err;
    }
    rc = EVP_DigestUpdate(md_ctx, msg, sizeof(msg));
    if (rc != 1) {
        rc = -1;
        goto err;
    }
    rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
    if (rc != 1) {
        rc = -1;
        goto err;
    }
    EVP_MD_CTX_destroy(md_ctx);

    // Sign
    memset(sig, 0, sizeof(sig));
// Генерируем ключевую пару
BIO_printf(bio_out,
    "Generate NID_id_tc26_gost3410_2012_256 key pair with %s\n",
    SN_id_GostR3410_2001_CryptoPro_A_ParamSet);
if (st) {
    OPENSSL_ITEM attr_keypair[] = {
            {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
            {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
            {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
            {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
            {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_tc26_gost3410_2012_256;
```

```
    int parnid = NID_id_GostR3410_2001_CryptoPro_A_ParamSet;
    char *label = label_2012_256;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey = STORE_generate_key(st, attr_keypair, param_keypair);
    if (!pkey)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
   in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  } else {
    pectx = EVP_PKEY_CTX_new_id(NID_id_tc26_gost3410_2012_256, e
   );
    if (!pectx) {
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(pectx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(pectx, -1, -1,
   EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_GostR3410_2001_CryptoPro_A_ParamSet, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen(pectx, &pkey) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
```

```
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(pectx);
  }
  lcc_gost2012_key = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(pkey);
  if (lcc_gost2012_key->is_real_priv_key) {
    BIO_printf(bio_out, "Generated private key:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_value,
      lcc_gost2012_key->priv_key_len);
  } else {
    BIO_printf(bio_out, "Generated private key ID:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_id,
      sizeof(lcc_gost2012_key->priv_key_id));
  }
    // Create Sign context with ENGINE
    BIO_printf(bio_out, "Create sign context\n");
    ctx = EVP_PKEY_CTX_new(pkey, e);
    if (ctx) {
        // Sign hash data
        BIO_printf(bio_out, "Sign message hash\n");
        rc = EVP_PKEY_sign_init(ctx);
        if (rc == 1) {
            rc = EVP_PKEY_sign(ctx, sig, &sig_len, hash,
   hash_len);
            if (!rc) {
                rc = -1;
                goto err;
            }
            BIO_printf(bio_out, "Signature generated OK\n");
        } else {
            rc = -1;
            goto err;
        }
        EVP_PKEY_CTX_free(ctx);
    }

  // Verify
    // Create Verify context with ENGINE
    BIO_printf(bio_out, "Create verify context\n");
    ctx = EVP_PKEY_CTX_new(pkey, e);
    if (ctx) {
        // Verify signature
        BIO_printf(bio_out, "Verify message hash signature\n");
```

```
        rc = EVP_PKEY_verify_init(ctx);
        if (rc == 1) {
            rc = EVP_PKEY_verify(ctx, sig, sig_len, hash,
 hash_len);
            if (!rc) {
                rc = -1;
                goto err;
            }
            BIO_printf(bio_out, "Signature verified OK\n");
        } else {
            rc = -1;
            goto err;
        }
        EVP_PKEY_CTX_free(ctx);
    }
    EVP_PKEY_free(pkey);
  pkey = NULL;

  // Make message digest 3411-2012-512
    BIO_printf(bio_out,
    "Make NID_id_tc26_gost3411_2012_512 message digest\n");
    md = ENGINE_get_digest(e, NID_id_tc26_gost3411_2012_512);
    if(!md) {
        BIO_printf(bio_err,
      "Can't get digest NID_id_tc26_gost3411_2012_512\n");
        rc = -1;
        goto err;
    }
    md_ctx = EVP_MD_CTX_create();
    rc = EVP_DigestInit_ex(md_ctx, md, e);
    if (rc != 1) {
        rc = -1;
        goto err;
    }
    rc = EVP_DigestUpdate(md_ctx, msg, sizeof(msg));
    if (rc != 1) {
        rc = -1;
        goto err;
    }
    rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
    if (rc != 1) {
        rc = -1;
        goto err;
    }
```

```
    EVP_MD_CTX_destroy(md_ctx);

    // Sign
    memset(sig, 0, sizeof(sig));
  // Генерируем ключевую пару
    BIO_printf(bio_out,
    "Generate NID_id_tc26_gost3410_2012_512 key pair with %s\n",
    SN_id_tc26_gost_3410_2012_512_paramSetA);
  if (st) {
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
          {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
          {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
          {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_tc26_gost3410_2012_512;
    int parnid = NID_id_tc26_gost_3410_2012_512_paramSetA;
    char *label = label_2012_512;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey = STORE_generate_key(st, attr_keypair, param_keypair);
    if (!pkey)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
   in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  } else {
    pectx = EVP_PKEY_CTX_new_id(NID_id_tc26_gost3410_2012_512, e
   );
    if (!pectx) {
```

```
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(pectx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(pectx, -1, -1,
  EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_tc26_gost_3410_2012_512_paramSetA, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen(pectx, &pkey) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(pectx);
  }
  lcc_gost2012_key = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(pkey);
  if (lcc_gost2012_key->is_real_priv_key) {
    BIO_printf(bio_out, "Generated private key:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_value,
      lcc_gost2012_key->priv_key_len);
  } else {
    BIO_printf(bio_out, "Generated private key ID:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_id,
      sizeof(lcc_gost2012_key->priv_key_id));
  }
    // Create Sign context with ENGINE
    BIO_printf(bio_out, "Create sign context\n");
    ctx = EVP_PKEY_CTX_new(pkey, e);
    if (ctx) {
        // Sign hash data
        BIO_printf(bio_out, "Sign message hash\n");
        rc = EVP_PKEY_sign_init(ctx);
        if (rc == 1) {
            rc = EVP_PKEY_sign(ctx, sig, &sig_len, hash,
   hash_len);
```

```
            if (!rc) {
                rc = -1;
                goto err;
            }
            BIO_printf(bio_out, "Signature generated OK\n");
        } else {
            rc = -1;
            goto err;
        }
        EVP_PKEY_CTX_free(ctx);
    }

// Verify
    // Create Verify context with ENGINE
    BIO_printf(bio_out, "Create verify context\n");
    ctx = EVP_PKEY_CTX_new(pkey, e);
    if (ctx) {
        // Verify signature
        BIO_printf(bio_out, "Verify message hash signature\n");
        rc = EVP_PKEY_verify_init(ctx);
        if (rc == 1) {
            rc = EVP_PKEY_verify(ctx, sig, sig_len, hash,
 hash_len);
            if (!rc) {
                rc = -1;
                goto err;
            }
            BIO_printf(bio_out, "Signature verified OK\n");
        } else {
            rc = -1;
            goto err;
        }
        EVP_PKEY_CTX_free(ctx);
    }
    EVP_PKEY_free(pkey);
pkey = NULL;

if (st) {
    // Remove created token objects
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };
```

```
    attr_keypair[0].value = label_2001;
    attr_keypair[0].value_size = strlen(label_2001)+1;
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n");
      rc = -1;
      goto err;
    }
    rc = STORE_delete_public_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_public_key failed\n");
      rc = -1;
      goto err;
    }

    attr_keypair[0].value = label_2012_256;
    attr_keypair[0].value_size = strlen(label_2012_256)+1;
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n");
      rc = -1;
      goto err;
    }
    rc = STORE_delete_public_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_public_key failed\n");
      rc = -1;
      goto err;
    }

    attr_keypair[0].value = label_2012_512;
    attr_keypair[0].value_size = strlen(label_2012_512)+1;
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n");
      rc = -1;
      goto err;
    }
    rc = STORE_delete_public_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_public_key failed\n");
      rc = -1;
      goto err;
    }
```

```
  }

    BIO_printf(bio_err, "Free OpenSSL interface objects\n");

  if (st)
    STORE_free(st);
  st = NULL;
  if (e)
    ENGINE_free(e);
  e = NULL;

  if (bio_out)
    BIO_free(bio_out);

  BIO_printf(bio_err, "%s sign/verify test SUCCESS\n",
   engine_path);

  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  if (slot)
    OPENSSL_free(slot);
    rc = 0;
  return rc;

err:
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  if (slot)
    OPENSSL_free(slot);
  if (st)
    STORE_free(st);
    if (e)
        ENGINE_free(e);
  ERR_print_errors(bio_err);
  if (bio_out)
    BIO_free(bio_out);
  rc = -1;
    return rc;
}
```

5.5 VKO

Данный тест генерирует и сравнивает ключи обмена отправителя и получателя по алгоритмам VKO без диверсификации.

Листинг 5.5: lc_core_2012_vko.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/engine.h>
#include <openssl/store.h>
#include <openssl/err.h>
#include "lc_keypair_2012.h"
#include "lc_defs_2012.h"
#include "getopt.h"

void bio_print_hex(BIO *bio, unsigned char *buf, size_t len) {
    size_t i;
    for (i=0; i<len; i++)
        BIO_printf(bio, "%02X%c", buf[i],
        ((i+1)==len||!((i+1)%16))?'\n':' ');
}

BIO *bio_err = NULL;
BIO *bio_out = NULL;

// Тест демонстрирует применение алгоритмов VKO
// без диверсификации результирующего согласованного ключа.
int main(int argc, char **argv)
{
    int rc;
  ENGINE *e = NULL;
  STORE *st = NULL;
    EVP_PKEY *pkey_send = NULL;
    EVP_PKEY *pkey_recp = NULL;
    LC_GOST2012_KEYPAIR *gost2012_key_send = NULL;
    LC_GOST2012_KEYPAIR *gost2012_key_recp = NULL;
    static int param_buf;
    static unsigned int param_buf_len = 0;
    EVP_PKEY_CTX *ctx = NULL;
  // ТК 26 рекомендует использовать размер UKM,
```

```
  // равный 1/8 длины открытого ключа в байтах,
  // т.е. либо 8, либо 16 байтов.
    static unsigned char ukm[] = {
        0xfb, 0x2e, 0x4a, 0x12, 0x94, 0xda, 0xc7, 0x18,
    0x5b, 0xdc, 0xf2, 0x17, 0x34, 0x6a, 0x69, 0x2f,
    };
    unsigned char kek_send[256];
    unsigned int kek_send_len;
    unsigned char kek_recp[256];
    unsigned int kek_recp_len;
    EVP_PKEY_CTX *ctx_send = NULL;
    EVP_PKEY_CTX *ctx_recp = NULL;
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;
  char label_2001_send[] = "send_keypair_2001";
  char label_2001_recp[] = "recp_keypair_2001";
  char label_2012_256_send[] = "send_keypair_2012_256";
  char label_2012_256_recp[] = "recp_keypair_2012_256";
  char label_2012_512_send[] = "send_keypair_2012_512";
  char label_2012_512_recp[] = "recp_keypair_2012_512";

  while((c = getopt(argc, argv, "e:a:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
```

```
}

  // Error messages output
  if (bio_err == NULL)
  if ((bio_err = BIO_new(BIO_s_file())) != NULL)
    BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  // Standard output
  if (bio_out == NULL)
      if ((bio_out=BIO_new(BIO_s_file())) != NULL)
        BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  ENGINE_load_builtin_engines();
  BIO_printf(bio_out,
  "lc_core_2012_vko_derive test\n"
  "=================================================\n");

e = ENGINE_by_id("dynamic");
if (e)
{
      // Try to load other engine dynamically (calls ENGINE
 init())
  if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
    || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
  {
          BIO_printf(bio_err, "Loading ENGINE %s failed\n",
 engine_path);
          goto err;
  }
  if (api_path) {
    if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
      ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", api_path, 0)
      ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
 CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
      ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
    {
      BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
      goto err;
    }
    // Получаем из ENGINE интерфейс STORE.
    st = STORE_new_engine(e);
    if (!st) {
      BIO_printf(bio_err, "ENGINE has no STORE interface\n");
      rc = -1;
```

```
        goto err;
      }
    }
  }
  BIO_printf(bio_err, "Engine %s:0x%x loaded with id '%s'\n",
    engine_path, e, ENGINE_get_id(e));

  // 2001
    BIO_printf(bio_out,
    "Generate sender NID_id_GostR3410_2001 key pair\n"
    " with %s\n",
    SN_id_GostR3410_2001_CryptoPro_A_ParamSet);
  if (st) {
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
          {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
          {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
          {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_GostR3410_2001;
    int parnid = NID_id_GostR3410_2001_CryptoPro_A_ParamSet;
    char *label = label_2001_send;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey_send = STORE_generate_key(st, attr_keypair,
   param_keypair);
    if (!pkey_send)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
   in store\n",
```

```
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  } else {
    ctx = EVP_PKEY_CTX_new_id(NID_id_GostR3410_2001, e);
    if (!ctx) {
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(ctx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx, -1, -1,
   EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_GostR3410_2001_CryptoPro_A_ParamSet, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen(ctx, &pkey_send) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(ctx);
  }
  if (!pkey_send) {
    rc = -1;
    goto err;
  }
    // Get GOST keypair interface data
    gost2012_key_send = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(
   pkey_send);

    BIO_printf(bio_out,
    "Generate recipient NID_id_GostR3410_2001 key pair\n"
    " with %s\n",
    SN_id_GostR3410_2001_CryptoPro_A_ParamSet);
  if (st) {
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
```

```
    };
    OPENSSL_ITEM param_keypair[] = {
            {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
            {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
            {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_GostR3410_2001;
    int parnid = NID_id_GostR3410_2001_CryptoPro_A_ParamSet;
    char *label = label_2001_recp;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey_recp = STORE_generate_key(st, attr_keypair,
   param_keypair);
    if (!pkey_recp)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
   in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  } else {
    ctx = EVP_PKEY_CTX_new_id(NID_id_GostR3410_2001, e);
    if (!ctx) {
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(ctx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx, -1, -1,
   EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_GostR3410_2001_CryptoPro_A_ParamSet, NULL);
```

```
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen(ctx, &pkey_recp) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(ctx);
  }
    if (!pkey_recp) {
        rc = -1;
        goto err;
    }
    // Get GOST keypair interface data
    gost2012_key_recp = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(
   pkey_recp);

    BIO_printf(bio_out, "Derive sender key encryption key\n");
    ctx_send = EVP_PKEY_CTX_new(pkey_send, e);
    if (!ctx_send) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_new failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_init(ctx_send);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_init failed\n");
        goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx_send, -1,
    EVP_PKEY_OP_DERIVE, EVP_PKEY_CTRL_SET_IV,
    EVP_PKEY_size(pkey_recp)/8, ukm);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_set_peer(ctx_send, pkey_recp);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_set_peer failed\n")
   ;
        goto err;
    }
```

```
// При использовании lc_p11_core_2012 генерируется сессионный
// ключ согласования, поэтому на токене он не сохраняется.
  rc = EVP_PKEY_derive(ctx_send, kek_send, &kek_send_len);
  if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_derive failed\n");
      goto err;
  }
  EVP_PKEY_CTX_free(ctx_send);
  if (kek_send_len != 32) {
      BIO_printf(bio_err, "Invalid kek_send_len = %d\n",
 kek_send_len);
      goto err;
  }

  BIO_printf(bio_out, "Derive recipient key encryption key\n")
 ;
  ctx_recp = EVP_PKEY_CTX_new(pkey_recp, e);
  if (!ctx_recp) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_new failed\n");
      goto err;
  }
  rc = EVP_PKEY_derive_init(ctx_recp);
  if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_derive_init failed\n");
      goto err;
  }
  rc = EVP_PKEY_CTX_ctrl(ctx_recp, -1,
  EVP_PKEY_OP_DERIVE, EVP_PKEY_CTRL_SET_IV,
  EVP_PKEY_size(pkey_recp)/8, ukm);
  if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      goto err;
  }
  rc = EVP_PKEY_derive_set_peer(ctx_recp, pkey_send);
  if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_derive_set_peer failed\n")
 ;
      goto err;
  }
// При использовании lc_p11_core_2012 генерируется сессионный
// ключ согласования, поэтому на токене он не сохраняется.
  rc = EVP_PKEY_derive(ctx_recp, kek_recp, &kek_recp_len);
  if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_derive failed\n");
```

```
        goto err;
    }
    EVP_PKEY_CTX_free(ctx_recp);
    if (kek_recp_len != 32) {
        BIO_printf(bio_err, "Invalid kek_recp_len = %d\n",
  kek_recp_len);
        goto err;
    }
  BIO_printf(bio_err, "kek_send:\n");
  bio_print_hex(bio_err, kek_send, 32);
  BIO_printf(bio_err, "kek_recp:\n");
  bio_print_hex(bio_err, kek_recp, 32);
    if (memcmp(kek_send, kek_recp, 32) != 0) {
        BIO_printf(bio_err, "kek_send and kek_recp are different
  \n");
        goto err;
    }
    BIO_printf(bio_out,
    "Sender and recipient key encryption keys are equal\n"
    "================================================\n");

  // 2012-256
    BIO_printf(bio_out,
    "Generate sender NID_id_tc26_gost3410_2012_256 key pair\n"
    " with %s\n",
    SN_id_GostR3410_2001_CryptoPro_A_ParamSet);
  if (st) {
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
          {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
          {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
          {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_tc26_gost3410_2012_256;
    int parnid = NID_id_GostR3410_2001_CryptoPro_A_ParamSet;
    char *label = label_2012_256_send;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
```

```
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey_send = STORE_generate_key(st, attr_keypair,
   param_keypair);
    if (!pkey_send)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
   in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  } else {
    ctx = EVP_PKEY_CTX_new_id(NID_id_tc26_gost3410_2012_256, e);
    if (!ctx) {
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(ctx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx, -1, -1,
   EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_GostR3410_2001_CryptoPro_A_ParamSet, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen(ctx, &pkey_send) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(ctx);
  }
    if (!pkey_send) {
        rc = -1;
```

```
        goto err;
    }
    // Get GOST keypair interface data
    gost2012_key_send = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(
  pkey_send);

    BIO_printf(bio_out,
    "Generate recipient NID_id_tc26_gost3410_2012_256 key pair\n
  "
    " with %s\n",
    SN_id_GostR3410_2001_CryptoPro_A_ParamSet);
  if (st) {
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
          {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
          {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
          {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_tc26_gost3410_2012_256;
    int parnid = NID_id_GostR3410_2001_CryptoPro_A_ParamSet;
    char *label = label_2012_256_recp;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey_recp = STORE_generate_key(st, attr_keypair,
  param_keypair);
    if (!pkey_recp)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
  in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
```

```
  } else {
    ctx = EVP_PKEY_CTX_new_id(NID_id_tc26_gost3410_2012_256, e);
    if (!ctx) {
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(ctx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx, -1, -1,
   EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_GostR3410_2001_CryptoPro_A_ParamSet, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen(ctx, &pkey_recp) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(ctx);
  }
    if (!pkey_recp) {
        rc = -1;
        goto err;
    }
    // Get GOST keypair interface data
    gost2012_key_recp = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(
   pkey_recp);

    BIO_printf(bio_out, "Derive sender key encryption key\n");
    ctx_send = EVP_PKEY_CTX_new(pkey_send, e);
    if (!ctx_send) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_new failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_init(ctx_send);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_init failed\n");
```

```
        goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx_send, -1,
    EVP_PKEY_OP_DERIVE, EVP_PKEY_CTRL_SET_IV,
    EVP_PKEY_size(pkey_recp)/8, ukm);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_set_peer(ctx_send, pkey_recp);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_set_peer failed\n")
   ;
        goto err;
    }
  // При использовании lc_p11_core_2012 генерируется сессионный
  // ключ согласования, поэтому на токене он не сохраняется.
    rc = EVP_PKEY_derive(ctx_send, kek_send, &kek_send_len);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive failed\n");
        goto err;
    }
    EVP_PKEY_CTX_free(ctx_send);
    if (kek_send_len != 32) {
        BIO_printf(bio_err, "Invalid kek_send_len = %d\n",
   kek_send_len);
        goto err;
    }

    BIO_printf(bio_out, "Derive recipient key encryption key\n")
   ;
    ctx_recp = EVP_PKEY_CTX_new(pkey_recp, e);
    if (!ctx_recp) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_new failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_init(ctx_recp);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_init failed\n");
        goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx_recp, -1,
    EVP_PKEY_OP_DERIVE, EVP_PKEY_CTRL_SET_IV,
    EVP_PKEY_size(pkey_recp)/8, ukm);
```

```
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_set_peer(ctx_recp, pkey_send);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_set_peer failed\n")
  ;
        goto err;
    }
  // При использовании lc_p11_core_2012 генерируется сессионный
  // ключ согласования, поэтому на токене он не сохраняется.
    rc = EVP_PKEY_derive(ctx_recp, kek_recp, &kek_recp_len);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive failed\n");
        goto err;
    }
    EVP_PKEY_CTX_free(ctx_recp);
    if (kek_recp_len != 32) {
        BIO_printf(bio_err, "Invalid kek_recp_len = %d\n",
   kek_recp_len);
        goto err;
    }
    if (memcmp(kek_send, kek_recp, 32) != 0) {
        BIO_printf(bio_err, "kek_send and kek_recp are different
   \n");
        goto err;
    }
    BIO_printf(bio_out,
    "Sender and recipient key encryption keys are equal\n"
    "================================================\n");

  // 2012-512
    BIO_printf(bio_out,
    "Generate sender NID_id_tc26_gost3410_2012_512 key pair\n"
    " with %s\n",
    SN_id_tc26_gost_3410_2012_512_paramSetA);
  if (st) {
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
          {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
```

```
            {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
            {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_tc26_gost3410_2012_512;
    int parnid = NID_id_tc26_gost_3410_2012_512_paramSetA;
    char *label = label_2012_512_send;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey_send = STORE_generate_key(st, attr_keypair,
  param_keypair);
    if (!pkey_send)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
  in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  } else {
    ctx = EVP_PKEY_CTX_new_id(NID_id_tc26_gost3410_2012_512, e);
    if (!ctx) {
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(ctx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx, -1, -1,
  EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_tc26_gost_3410_2012_512_paramSetA, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
```

```
      goto err;
    }
    if (EVP_PKEY_keygen(ctx, &pkey_send) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(ctx);
  }
    if (!pkey_send) {
        rc = -1;
        goto err;
    }
    // Get GOST keypair interface data
    gost2012_key_send = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(
   pkey_send);

    BIO_printf(bio_out,
    "Generate recipient NID_id_tc26_gost3410_2012_512 key pair\n
   "
    " with %s\n",
    SN_id_tc26_gost_3410_2012_512_paramSetA);
  if (st) {
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
          {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
          {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
          {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_tc26_gost3410_2012_512;
    int parnid = NID_id_tc26_gost_3410_2012_512_paramSetA;
    char *label = label_2012_512_recp;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
    pkey_recp = STORE_generate_key(st, attr_keypair,
   param_keypair);
```

```
    if (!pkey_recp)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = -1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
   in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  } else {
    ctx = EVP_PKEY_CTX_new_id(NID_id_tc26_gost3410_2012_512, e);
    if (!ctx) {
      fprintf(stderr, "EVP_PKEY_CTX_new_id failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen_init(ctx) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen_init failed\n");
      rc = -1;
      goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx, -1, -1,
   EVP_PKEY_CTRL_GOST2012_PARAMSET,
      NID_id_tc26_gost_3410_2012_512_paramSetA, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
      rc = -1;
      goto err;
    }
    if (EVP_PKEY_keygen(ctx, &pkey_recp) <= 0) {
      fprintf(stderr, "EVP_PKEY_keygen failed\n");
      rc = -1;
      goto err;
    }
    EVP_PKEY_CTX_free(ctx);
  }
    if (!pkey_recp) {
        rc = -1;
        goto err;
    }
    // Get GOST keypair interface data
    gost2012_key_recp = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(
```

```
    pkey_recp);

    BIO_printf(bio_out, "Derive sender key encryption key\n");
    ctx_send = EVP_PKEY_CTX_new(pkey_send, e);
    if (!ctx_send) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_new failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_init(ctx_send);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_init failed\n");
        goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx_send, -1,
    EVP_PKEY_OP_DERIVE, EVP_PKEY_CTRL_SET_IV,
    EVP_PKEY_size(pkey_recp)/8, ukm);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_set_peer(ctx_send, pkey_recp);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_set_peer failed\n")
    ;
        goto err;
    }
   // При использовании lc_p11_core_2012 генерируется сессионный
   // ключ согласования, поэтому на токене он не сохраняется.
    rc = EVP_PKEY_derive(ctx_send, kek_send, &kek_send_len);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive failed\n");
        goto err;
    }
    EVP_PKEY_CTX_free(ctx_send);
    if (kek_send_len != 32) {
        BIO_printf(bio_err, "Invalid kek_send_len = %d\n",
    kek_send_len);
        goto err;
    }

    BIO_printf(bio_out, "Derive recipient key encryption key\n")
    ;
    ctx_recp = EVP_PKEY_CTX_new(pkey_recp, e);
    if (!ctx_recp) {
```

```
        BIO_printf(bio_err, "EVP_PKEY_CTX_new failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_init(ctx_recp);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_init failed\n");
        goto err;
    }
    rc = EVP_PKEY_CTX_ctrl(ctx_recp, -1,
    EVP_PKEY_OP_DERIVE, EVP_PKEY_CTRL_SET_IV,
    EVP_PKEY_size(pkey_recp)/8, ukm);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_CTX_ctrl failed\n");
        goto err;
    }
    rc = EVP_PKEY_derive_set_peer(ctx_recp, pkey_send);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive_set_peer failed\n")
  ;
        goto err;
    }
  // При использовании lc_p11_core_2012 генерируется сессионный
  // ключ согласования, поэтому на токене он не сохраняется.
    rc = EVP_PKEY_derive(ctx_recp, kek_recp, &kek_recp_len);
    if (rc != 1) {
        BIO_printf(bio_err, "EVP_PKEY_derive failed\n");
        goto err;
    }
    EVP_PKEY_CTX_free(ctx_recp);
    if (kek_recp_len != 32) {
        BIO_printf(bio_err, "Invalid kek_recp_len = %d\n",
  kek_recp_len);
        goto err;
    }
    if (memcmp(kek_send, kek_recp, 32) != 0) {
        BIO_printf(bio_err, "kek_send and kek_recp are different
  \n");
        goto err;
    }
    BIO_printf(bio_out,
    "Sender and recipient key encryption keys are equal\n"
    "=================================================\n");
  if (st) {
    // Remove created token objects
```

```
    OPENSSL_ITEM attr_keypair[] = {
            {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
            {STORE_ATTR_END, NULL, 0, NULL}
    };

    attr_keypair[0].value = label_2001_send;
    attr_keypair[0].value_size = strlen(label_2001_send)+1;
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n");
      rc = -1;
      goto err;
    }
    rc = STORE_delete_public_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_public_key failed\n");
      rc = -1;
      goto err;
    }
    attr_keypair[0].value = label_2001_recp;
    attr_keypair[0].value_size = strlen(label_2001_recp)+1;
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n");
      rc = -1;
      goto err;
    }
    rc = STORE_delete_public_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_public_key failed\n");
      rc = -1;
      goto err;
    }

    attr_keypair[0].value = label_2012_256_send;
    attr_keypair[0].value_size = strlen(label_2012_256_send)+1;
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n");
      rc = -1;
      goto err;
    }
    rc = STORE_delete_public_key(st, attr_keypair, NULL);
    if (rc != 1) {
```

```
    BIO_printf(bio_err, "STORE_delete_public_key failed\n");
    rc = -1;
    goto err;
  }
  attr_keypair[0].value = label_2012_256_recp;
  attr_keypair[0].value_size = strlen(label_2012_256_recp)+1;
  rc = STORE_delete_private_key(st, attr_keypair, NULL);
  if (rc != 1) {
    BIO_printf(bio_err, "STORE_delete_private_key failed\n");
    rc = -1;
    goto err;
  }
  rc = STORE_delete_public_key(st, attr_keypair, NULL);
  if (rc != 1) {
    BIO_printf(bio_err, "STORE_delete_public_key failed\n");
    rc = -1;
    goto err;
  }

  attr_keypair[0].value = label_2012_512_send;
  attr_keypair[0].value_size = strlen(label_2012_512_send)+1;
  rc = STORE_delete_private_key(st, attr_keypair, NULL);
  if (rc != 1) {
    BIO_printf(bio_err, "STORE_delete_private_key failed\n");
    rc = -1;
    goto err;
  }
  rc = STORE_delete_public_key(st, attr_keypair, NULL);
  if (rc != 1) {
    BIO_printf(bio_err, "STORE_delete_public_key failed\n");
    rc = -1;
    goto err;
  }
  attr_keypair[0].value = label_2012_512_recp;
  attr_keypair[0].value_size = strlen(label_2012_512_recp)+1;
  rc = STORE_delete_private_key(st, attr_keypair, NULL);
  if (rc != 1) {
    BIO_printf(bio_err, "STORE_delete_private_key failed\n");
    rc = -1;
    goto err;
  }
  rc = STORE_delete_public_key(st, attr_keypair, NULL);
  if (rc != 1) {
    BIO_printf(bio_err, "STORE_delete_public_key failed\n");
```

```
        rc = -1;
        goto err;
      }
    }

    BIO_printf(bio_out, "lc_core_2012_vko_derive test SUCCESS\n");

    if (pkey_send)
        EVP_PKEY_free(pkey_send);
    if (pkey_recp)
        EVP_PKEY_free(pkey_recp);
  if (st)
    STORE_free(st);
  st = NULL;
  if (e)
    ENGINE_free(e);
  e = NULL;

  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  if (bio_out)
    BIO_free(bio_out);
    rc = 0;
    return rc;

err:
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
    if (pkey_send)
        EVP_PKEY_free(pkey_send);
    if (pkey_recp)
        EVP_PKEY_free(pkey_recp);
  if (st)
    STORE_free(st);
    if (e)
        ENGINE_free(e);
  ERR_print_errors(bio_err);
  if (bio_out)
    BIO_free(bio_out);
```

```
  rc = -1;
    return rc;
}
```

5.6 X509

Загрузка и проверка сертификата:

Листинг 5.6: lc_core_2012_x509.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/bio.h>
#include <openssl/engine.h>
#include <openssl/crypto.h>
#include <openssl/pem.h>
#include <openssl/x509.h>
#include <openssl/x509_vfy.h>
#include <openssl/err.h>
#include "getopt.h"

int main(int argc, char **argv)
{
  ENGINE *e = NULL;
  const char ca_bundlestr[] = "131030/cms-sign2/cms2CA-2012/
   cacert.pem";
  const char cert_filestr[] = "131030/cms-sign2/U_cms_2/cert.pem
   ";
  BIO               *certbio = NULL;
  BIO                *outbio = NULL;
  BIO                *bio_err = NULL;
  X509          *error_cert = NULL;
  X509                *cert = NULL;
  X509_NAME     *certsubject = NULL;
  X509_STORE          *store = NULL;
  X509_STORE_CTX  *vrfy_ctx = NULL;
  int rc;
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
```

```
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;

  while((c = getopt(argc, argv, "e:a:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

  // ------------------------------------------------------------
   *
  // These function calls initialize openssl for correct work.
   *
  // ------------------------------------------------------------
   *
  OpenSSL_add_all_algorithms();
  ERR_load_BIO_strings();
  ERR_load_crypto_strings();
  // ------------------------------------------------------------
   *
  // Create the Input/Output BIO's.
   *
  // ------------------------------------------------------------
   *
  certbio = BIO_new(BIO_s_file());
  outbio  = BIO_new_fp(stdout, BIO_NOCLOSE);
  bio_err  = BIO_new_fp(stderr, BIO_NOCLOSE);

    BIO_printf(outbio, "lc_core_2012_x509 test\n");
    // Builtin engines should be loaded first
    // to access builtin "dynamic" engine then.
```

```
    ENGINE_load_builtin_engines();
  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load other engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
          BIO_printf(bio_err, "Loading ENGINE %s failed\n",
   engine_path);
          goto err;
    }
    if (api_path) {
      if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
   CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
      {
        BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
        goto err;
      }
    }
  }
  BIO_printf(bio_err, "Engine %s:0x%x loaded with id '%s'\n",
    engine_path, e, ENGINE_get_id(e));


  // ------------------------------------------------------------
   *
  // Initialize the global certificate validation store object.
   *
  // ------------------------------------------------------------
   *
  if (!(store=X509_STORE_new())) {
     BIO_printf(outbio, "Error creating X509_STORE_CTX object\n"
   );
   exit(-1);
  }

  // ------------------------------------------------------------
   *
  // Create the context structure for the validation operation.
```

```
 *
// ------------------------------------------------------------
 *
vrfy_ctx = X509_STORE_CTX_new();

// ------------------------------------------------------------
 *
// Load the certificate and cacert chain from file (PEM).
 *
// ------------------------------------------------------------
 *
BIO_printf(outbio, "Load the certificate from file (PEM)\n");
rc = BIO_read_filename(certbio, cert_filestr);
if (! (cert = PEM_read_bio_X509(certbio, NULL, 0, NULL))) {
  BIO_printf(outbio, "Error loading cert into memory\n");
  exit(-1);
}

BIO_printf(outbio, "Load cacert chain from file (PEM)\n");
rc = X509_STORE_load_locations(store, ca_bundlestr, NULL);
if (rc != 1) {
  BIO_printf(outbio, "Error loading CA cert or chain file\n");
  exit(-1);
}

BIO_printf(outbio,
  "Verify the certificate against cacert chain\n");
// ------------------------------------------------------------
 *
// Initialize the ctx structure for a verification operation:
 *
// Set the trusted cert store, the unvalidated cert, and any
 *
// potential certs that could be needed (here we set it NULL)
 *
// ------------------------------------------------------------
 *
X509_STORE_CTX_init(vrfy_ctx, store, cert, NULL);

// ------------------------------------------------------------
 *
// Check the complete cert chain can be build and validated.
 *
// Returns 1 on success, 0 on verification failures, and -1
```

```
    *
  // for trouble with the ctx object (i.e. missing certificate)
    *
  // ------------------------------------------------------------
    *
  rc = X509_verify_cert(vrfy_ctx);
  BIO_printf(outbio, "Verification return code: %d\n", rc);

  if(rc == 0 || rc == 1)
  BIO_printf(outbio, "Verification result text: %s\n",
         X509_verify_cert_error_string(vrfy_ctx->error));

  // ------------------------------------------------------------
    *
  // The error handling below shows how to get failure details
    *
  // from the offending certificate.
    *
  // ------------------------------------------------------------
    *
  if(rc == 0) {
    //  get the offending certificate causing the failure *
    error_cert  = X509_STORE_CTX_get_current_cert(vrfy_ctx);
    certsubject = X509_NAME_new();
    certsubject = X509_get_subject_name(error_cert);
    BIO_printf(outbio, "Verification failed cert:\n");
    X509_NAME_print_ex(outbio, certsubject, 0, XN_FLAG_MULTILINE
   );
    BIO_printf(outbio, "\n");
  }

    BIO_printf(outbio, "lc_core_2012_x509 test SUCCESS\n");

err:
  // ------------------------------------------------------------
    *
  // Free up all structures
    *
  // ------------------------------------------------------------
    *
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
```

```
  X509_STORE_CTX_free(vrfy_ctx);
  X509_STORE_free(store);
  X509_free(cert);
  ERR_print_errors(bio_err);
  BIO_free_all(certbio);
  BIO_free_all(outbio);
  BIO_free_all(bio_err);
    ENGINE_free(e);

  return 0;
}
```

5.7 PKCS7

5.7.1 Зашифрование

Листинг 5.7: lc_core_2012_pkcs7_encrypt.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <sys/timeb.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/evp.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/pkcs7.h>
#include <openssl/engine.h>
#include <openssl/err.h>
#include "getopt.h"

BIO *bio_err = NULL;
BIO *bio_out = NULL;
ENGINE *e = NULL;
BIO *cert = NULL;
X509 *x = NULL;
char *cert_path_2012_256_XA_A =
"131030/cms_encr/U_cms_enc_gost2012_256_XA_1.2.643.2.2.31.1/cert
   .pem";
char *cert_path_2012_256_XB_Z =
```

```
"131030/cms_encr/U_cms_enc_gost2012_256_XB_1.2.643.7.1.2.5.1.1/
    cert.pem";
char *cert_path_2012_512_B_Z =
"131030/cms_encr/U_cms_enc_gost2012_512_B_1.2.643.7.1.2.5.1.1/
    cert.pem";
const char* param_28147_A = "id-Gost28147-89-CryptoPro-A-
    ParamSet";
const char* param_28147_Z = "id-tc26-gost-28147-89-param-A";
char *plain_path = "131030/cms_encr/encrypt_normalized.dat";
char *p7_2012_256_XA_A_path = "p7-2012-256-XA-encr-A.der";
char *p7_2012_256_XB_Z_path = "p7-2012-256-XB-encr-Z.der";
char *p7_2012_512_B_Z_path = "p7-2012-512-B-encr-Z.der";
STACK_OF(X509) *certs = NULL;
int rc;

int do_encrypt(char *p7_path)
{
    const EVP_CIPHER *c;
  STACK_OF( X509 ) *sk_certs = SKM_sk_new_null( X509 );
  BIO *in = NULL,
    *out = NULL;
  PKCS7 *p7 = NULL;

  c = ENGINE_get_cipher(e, NID_id_Gost28147_89);
    if(!c) {
        BIO_printf(bio_err,
      "Can't get cipher NID_id_Gost28147_89\n");
        goto err;
    }
  sk_X509_push(sk_certs, x);

  // Open content to encrypt
  in = BIO_new_file(plain_path, "r");
  if (!in) {
    goto err;
  }
  // Encrypt content
  p7 = PKCS7_encrypt(sk_certs, in, c, PKCS7_BINARY);
  if (!p7) {
    goto err;
  }
  BIO_free(in);

  out = BIO_new_file(p7_path, "wb");
```

```
  if (!out)
    goto err;

  // Write out DER message
  if ( 0 >= i2d_PKCS7_bio( out, p7 ) )
  {
    BIO_printf(bio_err,
      "i2d_PKCS7_bio() failed\n");
    goto err;
  }
  BIO_free(out);
  PKCS7_free(p7);
  BIO_printf(bio_out,
    "Message encrypted successfully\n");
  sk_X509_free(sk_certs);

  BIO_printf(bio_out,
    "Looks okay...\n");
  return 1;
err:
  BIO_printf(bio_err,
    "Error...\n");
  return 0;
}

int main( int argc, char **argv )
{
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;

  while((c = getopt(argc, argv, "e:a:"))!=-1){
      switch(c){
        case 'e':
```

```
      engine_path = BUF_strdup(optarg);
          break;
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

    // Error messages output
    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);
    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  BIO_printf(bio_out, "lc_core_2012_pkcs7_encrypt test START\n")
   ;

  ENGINE_load_builtin_engines();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load other engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
          BIO_printf(bio_err, "Loading ENGINE %s failed\n",
   engine_path);
          goto err;
    }
    if (api_path) {
      if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
   CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
      {
        BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
```

```
        goto err;
      }
    }
  }
  BIO_printf(bio_err, "Engine 0x%x loaded with id '%s'\n",
    e, ENGINE_get_id(e));

  // PARAMS_GOST28147 задается в качестве умалчиваемого параметр
   а
  // для шифрования данных, ключа и вычисления имитовставки,
  // если в контексте шифрования не задан соответствующий
   paramNID.
  rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            param_28147_A, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
   ;
    rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Load X.509 certificate from PEM file\n"
   );
    if ((cert = BIO_new(BIO_s_file())) == NULL)
    {
        rc = -1;
        goto err;
    }
    if (BIO_read_filename(cert, cert_path_2012_256_XA_A) <= 0)
    {
        BIO_printf(bio_err, "Error opening %s\n",
      cert_path_2012_256_XA_A);
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Create X509 interface object\n");
  x = PEM_read_bio_X509(cert, &x, NULL, NULL);
  if (!x) {
        BIO_printf(bio_err, "PEM_read_bio_X509 failed\n");
        rc = -1;
        goto err;
  }
    BIO_printf(bio_out, "Print certificate\n");
    X509_print(bio_out, x);
```

```
  rc = do_encrypt(p7_2012_256_XA_A_path);
  if (!rc) {
    BIO_printf(bio_err, "do_encrypt failed\n");
    rc = -1;
    goto err;
  } else {
    BIO_printf(bio_out, "do_encrypt(%s) SUCCESS\n",
      p7_2012_256_XA_A_path);
  }
  X509_free(x);
  x = NULL;

  rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            param_28147_Z, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
   ;
    rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Load X.509 certificate from PEM file\n"
   );
    if ((cert = BIO_new(BIO_s_file())) == NULL)
    {
        rc = -1;
        goto err;
    }
    if (BIO_read_filename(cert, cert_path_2012_256_XB_Z) <= 0)
    {
        BIO_printf(bio_err, "Error opening %s\n",
      cert_path_2012_256_XB_Z);
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Create X509 interface object\n");
  x = PEM_read_bio_X509(cert, &x, NULL, NULL);
  if (!x) {
        BIO_printf(bio_err, "PEM_read_bio_X509 failed\n");
        rc = -1;
        goto err;
  }
    BIO_printf(bio_out, "Print certificate\n");
    X509_print(bio_out, x);
```

```
  rc = do_encrypt(p7_2012_256_XB_Z_path);
  if (!rc) {
    BIO_printf(bio_err, "do_encrypt failed\n");
    rc = -1;
    goto err;
  } else {
    BIO_printf(bio_out, "do_encrypt(%s) SUCCESS\n",
      p7_2012_256_XB_Z_path);
  }
  X509_free(x);
  x = NULL;

  rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            param_28147_Z, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
   ;
    rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Load X.509 certificate from PEM file\n"
   );
    if ((cert = BIO_new(BIO_s_file())) == NULL)
    {
        rc = -1;
        goto err;
    }
    if (BIO_read_filename(cert, cert_path_2012_512_B_Z) <= 0)
    {
        BIO_printf(bio_err, "Error opening %s\n",
      cert_path_2012_512_B_Z);
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Create X509 interface object\n");
  x = PEM_read_bio_X509(cert, &x, NULL, NULL);
  if (!x) {
        BIO_printf(bio_err, "PEM_read_bio_X509 failed\n");
        rc = -1;
        goto err;
  }
    BIO_printf(bio_out, "Print certificate\n");
    X509_print(bio_out, x);
```

```
  rc = do_encrypt(p7_2012_512_B_Z_path);
  if (!rc) {
    BIO_printf(bio_err, "do_encrypt failed\n");
    rc = -1;
    goto err;
  } else {
    BIO_printf(bio_out, "do_encrypt(%s) SUCCESS\n",
      p7_2012_512_B_Z_path);
  }
  X509_free(x);
  x = NULL;

  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  ENGINE_free(e);
  EVP_cleanup();
  ENGINE_cleanup();
  CRYPTO_cleanup_all_ex_data();
  if (bio_out)
    BIO_free(bio_out);
  if (bio_err)
    BIO_free(bio_err);

  ERR_free_strings();
  return 0;

err:
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  ERR_print_errors(bio_err);
  if (bio_out)
    BIO_free(bio_out);
  if (bio_err)
    BIO_free(bio_err);

  ERR_free_strings();
  return -1;
}
```

5.7.2 Расшифрование

Листинг 5.8: lc_core_2012_pkcs7_decrypt.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <sys/timeb.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/evp.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/pkcs7.h>
#include <openssl/engine.h>
#include <openssl/store.h>
#include <openssl/err.h>
#include "lc_keypair_2012.h"
#include "getopt.h"

BIO *bio_err = NULL;
BIO *bio_out = NULL;
ENGINE *e = NULL;
STORE *st = NULL;
BIO *bio_pkey = NULL;
char *pkey_path_2012_256_XA_A =
"131030/cms_encr/U_cms_enc_gost2012_256_XA_1.2.643.2.2.31.1/
   seckey.pem";
char *pkey_path_2012_256_XB_Z =
"131030/cms_encr/U_cms_enc_gost2012_256_XB_1.2.643.7.1.2.5.1.1/
   seckey.pem";
char *pkey_path_2012_512_B_Z =
"131030/cms_encr/U_cms_enc_gost2012_512_B_1.2.643.7.1.2.5.1.1/
   seckey.pem";
char *pkey_path_2012_512_A_ =
"131030/cms_encr/U_cms_enc_gost2012_512_A_/seckey.pem";
EVP_PKEY *pkey = NULL;
LC_GOST2012_KEYPAIR * lcc_gost2012_key = NULL;
BIO *cert = NULL;
char *cert_path_2012_256_XA_A =
"131030/cms_encr/U_cms_enc_gost2012_256_XA_1.2.643.2.2.31.1/cert
   .pem";
char *cert_path_2012_256_XB_Z =
```

```
"131030/cms_encr/U_cms_enc_gost2012_256_XB_1.2.643.7.1.2.5.1.1/
    cert.pem";
char *cert_path_2012_512_B_Z =
"131030/cms_encr/U_cms_enc_gost2012_512_B_1.2.643.7.1.2.5.1.1/
    cert.pem";
char *cert_path_2012_512_A_ =
"131030/cms_encr/U_cms_enc_gost2012_512_A_/cert.pem";
X509 *x = NULL;
const char* param_28147_A = "id-Gost28147-89-CryptoPro-A-
    ParamSet";
const char* param_28147_Z = "id-tc26-gost-28147-89-param-A";
char *plain_path = "131030/cms_encr/encrypt_normalized.dat";
char *p7_2012_256_XA_A_path =
"131030/cms_encr/cms_enc_gost2012_256_XA_1.2.643.2.2.31.1.der";
char *p7_2012_256_XB_Z_path =
"131030/cms_encr/cms_enc_gost2012_256_XB_1.2.643.7.1.2.5.1.1.der
    ";
//char *p7_2012_512_B_Z_path =
//"131030/cms_encr/cms_enc_gost2012_512_B_1.2.643.7.1.2.5.1.1.
    der";
char *p7_2012_512_A_path = "131030/cms_encr/
    cms_enc_gost2012_512_A_.der";
//char *p7_2012_256_XA_A_path = "p7-2012-256-XA-encr-A.der";
//char *p7_2012_256_XB_Z_path = "p7-2012-256-XB-encr-Z.der";
char *p7_2012_512_B_Z_path = "p7-2012-512-B-encr-Z.der";
int rc;

int do_decrypt(char *p7_path)
{
  STACK_OF( PKCS7_RECIP_INFO )* recips = NULL;
  BIO *in = NULL,
    *out = NULL;
  PKCS7 *p7 = NULL;
  int i = 0, k = 0;
  unsigned char plain[1024];
  size_t plain_len = 0;
  unsigned char decrypted[1024];
  size_t decrypted_len = 0;

  // Open encrypted content
  in = BIO_new_file(p7_path, "rb");
  if (!in)
      goto err;
  if ( 0 == ( p7 = d2i_PKCS7_bio( in, 0 ) ) )
```

```
{
  BIO_printf(bio_err,
    "d2i_PKCS7_bio() failed\n");
  goto err;
}
BIO_free(in);

if ( !PKCS7_type_is_enveloped( p7 ) )
{
  BIO_printf(bio_err,
    "emout.der is not an enveloped message\n");
  goto err;
}

if ( NULL == ( recips = p7->d.enveloped->recipientinfo ) )
{
  printf( "No recipients' info found\n");
  goto err;
}

for( i = 0; i < sk_PKCS7_RECIP_INFO_num( recips ); ++i )
{
  PKCS7_RECIP_INFO* ri =
    sk_PKCS7_RECIP_INFO_value( recips, i );
  ASN1_INTEGER *serial = X509_get_serialNumber(x);

  // По хорошему, сертификат нужно искать по его
  // идентификационной информации в ri->issuer_and_serial.
  // Мы же просто проверим серийный номер сертификата
  // получателя для очистки совести.
  if (M_ASN1_INTEGER_cmp(ri->issuer_and_serial->serial,
      serial) != 0)
    continue;
  if ( NULL == ( out = BIO_new_file( "dmout.txt", "wb" ) ) )
  {
    BIO_printf(bio_err,
      "BIO_new_file( dmout.txt ) failed\n");
    goto err;
  }
  if ( 0 >= ( k = PKCS7_decrypt( p7, pkey, x, out, 0 ) ) )
  {
    BIO_printf(bio_err,
      "PKCS7_decrypt() failed\n");
    goto err;
```

```
    }
    BIO_free(out);
    break;
  }
  if ( i == sk_PKCS7_RECIP_INFO_num( recips ) )
  {
    BIO_printf(bio_err,
"Recipient's certificate and/or private key not found\n");
    goto err;
  }
  PKCS7_free(p7);

  if ((in = BIO_new(BIO_s_file())) == NULL)
    {
        goto err;
    }
    if (BIO_read_filename(in, plain_path) <= 0)
    {
        BIO_printf(bio_err,
      "Error opening %s\n",
      plain_path);
        goto err;
    }
  plain_len = BIO_read(in, plain, sizeof(plain));
  BIO_free(in);

  if ((in = BIO_new(BIO_s_file())) == NULL)
    {
        goto err;
    }
    if (BIO_read_filename(in, "dmout.txt") <= 0)
    {
        BIO_printf(bio_err,
      "Error opening %s\n",
      "dmout.txt");
        goto err;
    }
  decrypted_len = BIO_read(in, decrypted, sizeof(decrypted));
  BIO_free(in);

  if (plain_len != decrypted_len) {
        BIO_printf(bio_err, "Invalid decrypted text length\n");
    goto err;
  }
```

```
  if (memcmp(plain, decrypted, plain_len) != 0) {
        BIO_printf(bio_err, "Invalid decrypted text\n");
    goto err;
  }
  BIO_printf(bio_out,
    "Message decrypted successfully\n");

  EVP_PKEY_free(pkey);

  BIO_printf(bio_out,
    "Looks okay...\n");
    return 1;
err:
    BIO_printf(bio_err,
    "Error...\n");
    return 0;
}

int main( int argc, char **argv )
{
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;
  char label[] = "test_pkey";
  OPENSSL_ITEM attr_label[] = {
        {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
        {STORE_ATTR_END, NULL, 0, NULL}
  };

  while((c = getopt(argc, argv, "e:a:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
```

```
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }
    // Error messages output
    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);
    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  BIO_printf(bio_out, "lc_core_2012_pkcs7_decrypt test START\n")
   ;

  ENGINE_load_builtin_engines();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load other engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
            BIO_printf(bio_err, "Loading ENGINE %s failed\n",
   engine_path);
            goto err;
    }
    if (api_path) {
      if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
   CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
      {
        BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
        goto err;
      }
      // Получаем из ENGINE интерфейс STORE.
```

```
      st = STORE_new_engine(e);
      if (!st) {
        BIO_printf(bio_err, "ENGINE has no STORE interface\n");
        rc = -1;
        goto err;
      }
    }
  }
  BIO_printf(bio_err, "Engine 0x%x loaded with id '%s'\n",
    e, ENGINE_get_id(e));

  // PARAMS_GOST28147 влияет только на зашифрование данных.
  // Расшифрование производится с параметрами из структуры P7.
/*
  rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            param_28147_A, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
   ;
    rc = -1;
        goto err;
    }
*/
  BIO_printf(bio_err,
    "Import private key from non-encrypted PKCS#8 PEM file %s\n"
   ,
    pkey_path_2012_256_XA_A);
  bio_pkey = BIO_new_file(pkey_path_2012_256_XA_A, "rb");
  pkey = PEM_read_bio_PrivateKey(bio_pkey, NULL, NULL, NULL);
  if (!pkey) {
        BIO_printf(bio_err, "Can't read private key from PEM
   file\n");
        rc = -1;
      goto err;
  }
  if (st) {
    // Выгружаем закрытый ключ на токен через интерфейс STORE.
    attr_label[0].value = label;
    attr_label[0].value_size = strlen(label)+1;
    if ( 0 >= STORE_store_private_key( st, pkey, attr_label,
   NULL ) )
    {
      BIO_printf(bio_err, "STORE_store_private_key failed\n" );
      rc = -1;
```

```
        goto err;
      }
    }


    BIO_printf(bio_out, "Load X.509 certificate from PEM file\n"
    );
    if ((cert = BIO_new(BIO_s_file())) == NULL)
    {
        rc = -1;
        goto err;
    }
    if (BIO_read_filename(cert, cert_path_2012_256_XA_A) <= 0)
    {
        BIO_printf(bio_err, "Error opening %s\n",
      cert_path_2012_256_XA_A);
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Create X509 interface object\n");
  x = PEM_read_bio_X509(cert, &x, NULL, NULL);
  if (!x) {
        BIO_printf(bio_err, "PEM_read_bio_X509 failed\n");
        rc = -1;
        goto err;
  }
    BIO_printf(bio_out, "Print certificate\n");
    X509_print(bio_out, x);

  rc = do_decrypt(p7_2012_256_XA_A_path);
  if (!rc) {
    BIO_printf(bio_err, "do_decrypt(%s) failed\n",
      p7_2012_256_XA_A_path);
    rc = -1;
    goto err;
  } else {
    BIO_printf(bio_out, "do_decrypt(%s) SUCCESS\n",
      p7_2012_256_XA_A_path);
  }
  X509_free(x);
  x = NULL;

/*
  rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
```

```
            param_28147_Z, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
   ;
    rc = -1;
        goto err;
    }
*/
  BIO_printf(bio_err,
    "Import private key from non-encrypted PKCS#8 PEM file %s\n"
   ,
    pkey_path_2012_256_XB_Z);
  bio_pkey = BIO_new_file(pkey_path_2012_256_XB_Z, "rb");
  pkey = PEM_read_bio_PrivateKey(bio_pkey, NULL, NULL, NULL);
  if (!pkey) {
        BIO_printf(bio_err, "Can't read private key from PEM
   file\n");
        rc = -1;
      goto err;
  }
  if (st) {
    // Выгружаем закрытый ключ на токен через интерфейс STORE.
    attr_label[0].value = label;
    attr_label[0].value_size = strlen(label)+1;
    if ( 0 >= STORE_store_private_key( st, pkey, attr_label,
   NULL ) )
    {
      BIO_printf(bio_err, "STORE_store_private_key failed\n" );
      rc = -1;
      goto err;
    }
  }

    BIO_printf(bio_out, "Load X.509 certificate from PEM file\n"
   );
    if ((cert = BIO_new(BIO_s_file())) == NULL)
    {
        rc = -1;
        goto err;
    }
    if (BIO_read_filename(cert, cert_path_2012_256_XB_Z) <= 0)
    {
        BIO_printf(bio_err, "Error opening %s\n",
      cert_path_2012_256_XB_Z);
```

```
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Create X509 interface object\n");
  x = PEM_read_bio_X509(cert, &x, NULL, NULL);
  if (!x) {
        BIO_printf(bio_err, "PEM_read_bio_X509 failed\n");
        rc = -1;
        goto err;
  }
    BIO_printf(bio_out, "Print certificate\n");
    X509_print(bio_out, x);

  rc = do_decrypt(p7_2012_256_XB_Z_path);
  if (!rc) {
    BIO_printf(bio_err, "do_decrypt(%s) failed\n",
      p7_2012_256_XB_Z_path);
    rc = -1;
    goto err;
  } else {
    BIO_printf(bio_out, "do_decrypt(%s) SUCCESS\n",
      p7_2012_256_XB_Z_path);
  }
  X509_free(x);
  x = NULL;
/*
  rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            param_28147_Z, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
   ;
    rc = -1;
        goto err;
    }
*/
  BIO_printf(bio_err,
    "Import private key from non-encrypted PKCS#8 PEM file %s\n"
   ,
    pkey_path_2012_512_B_Z);
  bio_pkey = BIO_new_file(pkey_path_2012_512_B_Z, "rb");
  pkey = PEM_read_bio_PrivateKey(bio_pkey, NULL, NULL, NULL);
  if (!pkey) {
        BIO_printf(bio_err, "Can't read private key from PEM
   file\n");
```

```
        rc = -1;
      goto err;
  }
  if (st) {
    // Выгружаем закрытый ключ на токен через интерфейс STORE.
    attr_label[0].value = label;
    attr_label[0].value_size = strlen(label)+1;
    if ( 0 >= STORE_store_private_key( st, pkey, attr_label,
   NULL ) )
    {
      BIO_printf(bio_err, "STORE_store_private_key failed\n" );
      rc = -1;
      goto err;
    }
  }

    BIO_printf(bio_out, "Load X.509 certificate from PEM file\n"
   );
    if ((cert = BIO_new(BIO_s_file())) == NULL)
    {
        rc = -1;
        goto err;
    }
    if (BIO_read_filename(cert, cert_path_2012_512_B_Z) <= 0)
    {
        BIO_printf(bio_err, "Error opening %s\n",
      cert_path_2012_512_B_Z);
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Create X509 interface object\n");
  x = PEM_read_bio_X509(cert, &x, NULL, NULL);
  if (!x) {
        BIO_printf(bio_err, "PEM_read_bio_X509 failed\n");
        rc = -1;
        goto err;
  }
    BIO_printf(bio_out, "Print certificate\n");
    X509_print(bio_out, x);

  rc = do_decrypt(p7_2012_512_B_Z_path);
  if (!rc) {
    BIO_printf(bio_err, "do_decrypt(%s) failed\n",
      p7_2012_512_B_Z_path);
```

```
    rc = -1;
    goto err;
  } else {
    BIO_printf(bio_out, "do_decrypt(%s) SUCCESS\n",
      p7_2012_512_B_Z_path);
  }
  X509_free(x);
  x = NULL;

  if (st) {
    // Remove created token objects
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n");
      rc = -1;
      goto err;
    }
  }

  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  if (st)
    STORE_free(st);
  st = NULL;
  if (e)
    ENGINE_free(e);
  e = NULL;
  EVP_cleanup();
  ENGINE_cleanup();
  CRYPTO_cleanup_all_ex_data();
  if (bio_out)
    BIO_free(bio_out);
  if (bio_err)
    BIO_free(bio_err);
```

```
  ERR_free_strings();
  return 0;

err:
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  ERR_print_errors(bio_err);
  if (bio_out)
    BIO_free(bio_out);
  if (bio_err)
    BIO_free(bio_err);

  ERR_free_strings();
  return -1;
}
```

5.7.3 Подписывание

Листинг 5.9: lc_core_2012_pkcs7_sign.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <sys/timeb.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/pkcs7.h>
#include <openssl/engine.h>
#include <openssl/store.h>
#include <openssl/err.h>
#include "getopt.h"

#define PARAM_28147  "id-Gost28147-89-CryptoPro-A-ParamSet"

BIO *bio_err = NULL;
BIO *bio_out = NULL;

ENGINE *e = NULL;
STORE *st = NULL;
```

```
X509 *x = NULL;
EVP_PKEY *pkey = NULL;
STACK_OF(X509) *certs = NULL;
// GOST R34.10-2012-512
char *pkey_path = "131030/cms-sign2/U_cms_2/seckey.pem";
char *cert_path = "131030/cms-sign2/U_cms_2/cert.pem";
char *p7sig_path = "sig.der";
char *data_path = "data.bin";

int do_crypt()
{
  BIO *indata = NULL,
    *out = NULL;
  PKCS7 *p7 = NULL;
  int flags = PKCS7_BINARY|PKCS7_DETACHED;

  // Open content being signed
  // For detached signature we always use PKCS7_BINARY flag and
   "rb" mode
  indata = BIO_new_file("data.bin", "rb");
  if (!indata)
    goto err;
  // Sign content
  BIO_printf(bio_err, "PKCS7_sign\n");
  // If no certificate chain *stack_st_X509 included then
  // only signer certificate presented
  p7 = PKCS7_sign(x, pkey, certs, indata, flags);
  if (!p7)
    goto err;

  BIO_free(indata);

  out = BIO_new_file("sig.der", "wb");
  if (!out)
    goto err;

  if ( 0 >= i2d_PKCS7_bio( out, p7 ) )
  {
    BIO_printf(bio_err,
      "i2d_PKCS7_bio() failed\n");
    goto err;
  }

  BIO_free(out);
```

```
  PKCS7_free(p7);

  BIO_printf(bio_err,
    "PKCS7_sign success\n");

  BIO_printf(bio_err,
    "Looks okay...\n");
  return 1;
err:
  BIO_printf(bio_err,
    "Error...\n");
  return 0;
}

int main( int argc, char **argv )
{
  int         rc = 0;
  BIO *bio_pkey = NULL;
  BIO *cert = NULL;
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;
  char label[] = "test_pkey";
  OPENSSL_ITEM attr_label[] = {
        {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
        {STORE_ATTR_END, NULL, 0, NULL}
  };

  while((c = getopt(argc, argv, "e:a:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
```

```
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

    // Error messages output
    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);
    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  BIO_printf(bio_out, "lc_core_2012_pkcs7_sign test START\n");



    ENGINE_load_builtin_engines();
  OpenSSL_add_all_algorithms();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load other engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
            BIO_printf(bio_err, "Loading ENGINE %s failed\n",
   engine_path);
            goto err;
    }
    if (api_path) {
      if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
   CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
      {
        BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
```

```
        goto err;
      }
      // Получаем из ENGINE интерфейс STORE.
      st = STORE_new_engine(e);
      if (!st) {
        BIO_printf(bio_err, "ENGINE has no STORE interface\n");
        rc = -1;
        goto err;
      }
    }
  }
  BIO_printf(bio_err, "Engine 0x%x loaded with id '%s'\n",
    e, ENGINE_get_id(e));

    rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            PARAM_28147, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
   ;
        return -1;
    }
  BIO_printf(bio_err,
    "Import private key from non-encrypted PKCS#8 PEM file %s\n"
   ,
    pkey_path);
  bio_pkey = BIO_new_file(pkey_path, "rb");
  pkey = PEM_read_bio_PrivateKey(bio_pkey, NULL, NULL, NULL);
  if (!pkey) {
        BIO_printf(bio_err, "Can't read private key from PEM
   file\n");
        rc = -1;
      goto err;
  }
  if (st) {
    // Выгружаем закрытый ключ на токен через интерфейс STORE.
    attr_label[0].value = label;
    attr_label[0].value_size = strlen(label)+1;
    if ( 0 >= STORE_store_private_key( st, pkey, attr_label,
   NULL ) )
    {
      BIO_printf(bio_err, "STORE_store_private_key failed\n" );
      rc = -1;
      goto err;
    }
```

```
  }

    BIO_printf(bio_out, "Load X.509 certificate from PEM file\n"
   );
    if ((cert = BIO_new(BIO_s_file())) == NULL)
    {
        rc = -1;
        goto err;
    }
    if (BIO_read_filename(cert, cert_path) <= 0)
    {
        BIO_printf(bio_err, "Error opening %s\n", cert_path);
        rc = -1;
        goto err;
    }
    BIO_printf(bio_out, "Create X509 interface object\n");
  x = PEM_read_bio_X509(cert, &x, NULL, NULL);
  if (!x) {
        BIO_printf(bio_err, "PEM_read_bio_X509 failed\n");
        rc = -1;
        goto err;
  }
    BIO_printf(bio_out, "Print certificate\n");
    X509_print(bio_out, x);

  if (do_crypt()) {
    BIO_printf(bio_out, "lc_core_2012_pkcs7_sign test SUCCESS\n"
   );
    rc = 0;
  } else {
    BIO_printf(bio_err, "lc_core_2012_pkcs7_sign test FAILED\n")
   ;
    rc = -1;
  }

  if (st) {
    // Remove created token objects
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
```

```
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n");
      rc = -1;
      goto err;
    }
  }

  rc = 0;

  EVP_PKEY_free(pkey);
  X509_free(x);
  if (st)
    STORE_free(st);
  st = NULL;
  if (e)
    ENGINE_free(e);
  e = NULL;

  EVP_cleanup();
  ENGINE_cleanup();
  CRYPTO_cleanup_all_ex_data();

err:
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  ERR_print_errors(bio_err);

  if (bio_out)
    BIO_free(bio_out);
  if (bio_err)
    BIO_free(bio_err);

  ERR_free_strings();

  return rc;
}
```

5.7.4 Проверка подписи

Листинг 5.10: lc_core_2012_pkcs7_verify.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <sys/timeb.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/pkcs7.h>
#include <openssl/engine.h>
#include <openssl/err.h>
#include "getopt.h"

#define PARAM_28147  "id-Gost28147-89-CryptoPro-A-ParamSet"

BIO *bio_err = NULL;
BIO *bio_out = NULL;

ENGINE *e = NULL;
X509 *xca = NULL;
/*
char *cacert_path = "CryptoProCa.cer";
char *p7sig_path = "pdfsig.der";
char *data_path = "pdfdata.bin";
char *out_path = "pdfsigver.bin";
*/

char *cacert_path = "131030/cms-sign2/cms2CA-2012/cacert.der";
char *p7sig_path = "131030/cms-sign2/cms_signed2_1.der";
char *data_path = "131030/cms-sign2/cms_signed2.dat";
char *out_path = "sigver.bin";

int do_crypt()
{
  int flags = 0;
  PKCS7 *p7 = NULL;
  X509_STORE *st7 = NULL;
  BIO *in = NULL,
    *indata = NULL,
    *out = NULL;

  st7 = X509_STORE_new();
```

```
  if (!X509_STORE_add_cert(st7, xca))
    goto err;
  // PKCS7 signature verification
  if ( NULL == ( in = BIO_new_file( p7sig_path, "rb" ) ) )
  {
    BIO_printf(bio_err,
      "BIO_new_file( '%s' ) failed\n",
      p7sig_path);
    goto err;
  }
  if ( NULL == ( p7 = d2i_PKCS7_bio( in, NULL ) ) )
  {
    BIO_printf(bio_err,
      "d2i_PKCS7_bio() failed\n");
    goto err;
  }
  BIO_free(in);

  if ( !PKCS7_type_is_signed( p7 ) )
  {
    BIO_printf(bio_err,
      "'%s' has wrong format\n", p7sig_path );
    goto err;
  }

  if (PKCS7_is_detached(p7)) {
    // Input data (for PKCS7_detached only)
    // For detached PKCS7 we always use PKCS7_BINARY and "rb"
   mode
    if ( NULL == ( indata = BIO_new_file( data_path, "rb" ) ) )
    {
      BIO_printf(bio_err,
        "BIO_new_file( '%s' ) failed\n",
        data_path);
      goto err;
    }
  }

  if ( NULL == ( out = BIO_new_file( out_path, "wb" ) ) )
  {
    BIO_printf(bio_err,
      "BIO_new_file() failed\n");
    goto err;
  }
```

```
  // No other certificates or CRLs used in *stack_st_X509
  // (they would have high priority)
  if (0 == PKCS7_verify(p7, NULL, st7, indata, out, flags)) {
    BIO_printf(bio_err,
      "PKCS7_verify failed\n");
    goto err;
  }

  BIO_free(out);
  BIO_free(indata);
  PKCS7_free(p7);
  X509_STORE_free(st7);
  ENGINE_free(e);
  EVP_cleanup();
  ENGINE_cleanup();
  CRYPTO_cleanup_all_ex_data();
  BIO_printf(bio_out,
    "PKCS7_verify success\n");
  return 1;
err:
  BIO_printf(bio_err, "Error...\n");
  ERR_print_errors(bio_err);
  return 0;
}

int main( int argc, char **argv )
{
  int         rc;
    BIO *cacert = NULL;
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;
```

```
  while((c = getopt(argc, argv, "e:a:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

  ERR_load_crypto_strings();

    // Error messages output
    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stderr, BIO_NOCLOSE|BIO_FP_TEXT);
    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  BIO_printf(bio_out, "lc_core_2012_pkcs7_verify test START\n");

    ENGINE_load_dynamic();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load other engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
            BIO_printf(bio_err, "Loading ENGINE %s failed\n",
   engine_path);
            goto err;
    }
    if (api_path) {
      if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", api_path, 0)
```

```
        ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
    CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
      {
        BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
        goto err;
      }
    }
  }
  BIO_printf(bio_err, "Engine 0x%x loaded with id '%s'\n",
    e, ENGINE_get_id(e));

    rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            PARAM_28147, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
    ;
        return -1;
    }

    // Load trusted CA certificate
    BIO_printf(bio_out,
  "Loading X.509 trusted certificate from DER file %s\n",
    cacert_path);
    if ((cacert = BIO_new(BIO_s_file())) == NULL)
    {
        return -1;
    }
    if (BIO_read_filename(cacert, cacert_path) <= 0)
    {
        BIO_printf(bio_err,
      "Error opening %s\n",
      cacert_path);
        return -1;
    }
    BIO_printf(bio_out,
    "Create X509 interface object\n");
    xca = d2i_X509_bio(cacert, NULL);    // from DER
//  xca = PEM_read_bio_X509(cacert, &xca, NULL, NULL);  // from
   PEM
    if (!xca) {
        BIO_printf(bio_err,
    "Error creating X509 trusted certificate from DER file %s\n"
    ,
```

```
      cacert_path);
        return -1;
    }
  BIO_free(cacert);


  if (do_crypt()) {
    BIO_printf(bio_out, "lc_core_2012_pkcs7_verify test SUCCESS\
   n");
  } else {
    BIO_printf(bio_err, "lc_core_2012_pkcs7_verify test FAILED\n
   ");
  }

  X509_free(xca);

err:
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  EVP_cleanup();
  ENGINE_cleanup();
  CRYPTO_cleanup_all_ex_data();

  ERR_print_errors(bio_err);
  if (bio_out)
    BIO_free(bio_out);
  if (bio_err)
    BIO_free(bio_err);

  ERR_free_strings();

  return 0;
}
```

5.8 PKCS12

Поскольку для алгоритма ГОСТ Р34.11-2012 PKCS#12 пока не определен, то для HMAC при генерации ключа на пароле функцией PRF используется алгоритм ГОСТ Р34.11-94. Это не мешает паковать в "старый" контейнер PKCS#12 ключи и сертификаты, созданные по новым алгоритмам 2012.

Листинг 5.11: lc_core_2012_pkcs12.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/pkcs12.h>
#include <openssl/engine.h>
#include <openssl/err.h>
#include "getopt.h"

BIO *bio_err = NULL;
BIO *bio_out = NULL;

#define PARAM_28147  "id-tc26-gost-28147-89-param-A"

// Поскольку для алгоритма ГОСТ Р34.11-2012 PKCS#12 пока не опре
   делен,
// то для HMAC при генерации ключа на пароле функцией PRF исполь
   зуется
// алгоритм ГОСТ Р34.11-94.
// Это не мешает паковать в "старый" контейнер PKCS#12 ключи и с
   ертификаты,
// созданные по новым алгоритмам 2012.
int main(int argc, char **argv)
{
    int rc;
  ENGINE *e = NULL;
    EVP_PKEY *pkey = NULL;
    X509 *x = NULL;
  PKCS12 *p12 = NULL;
  int iter = 2048;
    int cert_pbe = NID_id_Gost28147_89;
    int key_pbe = NID_id_Gost28147_89;
  char *name = "server-512-111";
  char *inpass = "111";
  char *outpass = "1111";
    int keytype = 0; //KEY_SIG or KEY_EX - for MS
  STACK_OF(X509) *certs = NULL;
  const EVP_MD *macmd = NULL;
  BIO *bio_p12 = NULL;
```

```
  char *p12_path = "server-512-111.p12";
  char *p12_path_new = "p12_1111.pfx";
  BIO *bio_pkey = NULL;
  BIO *bio_cert = NULL;
  char *pkey_path = "server-512-111.pkey.pem";
  char *cert_path = "server-512-111.cert.pem";
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;

  while((c = getopt(argc, argv, "e:a:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

  if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    ENGINE_load_builtin_engines();
```

```
    BIO_printf(bio_out, "lc_core_2012_pkcs12 test\n");

  e = ENGINE_by_id("dynamic");
  if (e)
  {
        // Try to load other engine dynamically (calls ENGINE
   init())
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
            BIO_printf(bio_err, "Loading ENGINE %s failed\n",
   engine_path);
            goto err;
    }
    if (api_path) {
      if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
   CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
      {
        BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
        goto err;
      }
    }
  }
  BIO_printf(bio_err, "Engine %s:0x%x loaded with id '%s'\n",
    engine_path, e, ENGINE_get_id(e));

    rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            PARAM_28147, 0);
    if (!rc) {
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
   ;
        return -1;
    }

  // Читаем исходный тестовый контейнер,
  // изготовленный когда-то средствами LirSSL
  bio_p12 = BIO_new_file(p12_path, "rb");
  p12 = d2i_PKCS12_bio(bio_p12, NULL);
  BIO_free(bio_p12);

  // Распаковываем контейнер, зная, что
```

```
  // в нем хранится только одна ключевая пара.
  // Для этого используется упрощенный вариант
  // парсинга PKCS#12.
  rc = PKCS12_parse(p12, inpass, &pkey, &x,
       &certs);
  if (!rc) {
    goto err;
  }
  PKCS12_free(p12);
  p12 = NULL;

  // Если используется ENGINE lc_p11_core_2012, то
  // при импорте из P12 закрытый ключ и сертификат
  // не сохраняются на токене автоматически.
  // Для этого имеются функции STORE_store_private_key
  // и STORE_store_certificate.

  // Сохраняем закрытый ключ и сертификат в PEM-файлах.
  BIO_printf(bio_out, "Saving private key to '%s'\n", pkey_path)
    ;
  bio_pkey = BIO_new_file(pkey_path, "wb");
  PEM_write_bio_PrivateKey(
    bio_pkey, pkey, NULL, NULL, 0, NULL, NULL);
  BIO_free(bio_pkey);
  BIO_printf(bio_out, "Saving certificate to '%s'\n", cert_path)
    ;
  bio_cert = BIO_new_file(cert_path, "wb");
  PEM_write_bio_X509(bio_cert, x);
  BIO_free(bio_cert);

  // Приступаем к созданию нового контейнера PKCS#12
  // путем раздельной упаковки сертификата и закрытого ключа
  // с шифрованием в ASN.1-структуру PKCS#7 safes.
  // При экспорте мы пакуем здесь только сертификат,
  // соответствующий закрытому ключу, а остальную цепочку
  // сертификатов игнорируем. Поэтому экспортированный контейнер
  // может оказаться меньше исходного.
  // Заметим, что при создании нового контейнера используются
  // параметры ГОСТ 28147-89, заданные в контексте ENGINE
  // управляющей командой "PARAMS_GOST28147", поэтому они могут
  // отличаться от параметров исходного контейнера.

  BIO_printf(bio_out, "Create PKCS12\n");
  // PKCS12_create не умеет генерировать структуру дайджеста
```

```
  // для ГОСТ, поэтому мы ее добавим позже функцией
   PKCS12_set_mac.
  // А в данном вызове задаем для дайджеста количество итераций
   -1,
  // при котором структура дайджеста не строится.
  p12 = PKCS12_create(outpass, name, pkey, x, NULL,
    key_pbe, cert_pbe, iter, -1, keytype);
  if (!p12) {
    BIO_printf(bio_err, "PKCS12_create failed\n");
    goto err;
  }
  // Назначаем алгоритм хеширования для генерации дайджеста HMAC
  macmd = ENGINE_get_digest(e, NID_id_GostR3411_94);
  if (!macmd)
  {
    BIO_printf(bio_err, "ENGINE_get_digest failed\n");
    goto err;
  }
  // Добавляем дайджест HMAC для контроля целостности контейнера
  rc = PKCS12_set_mac(p12, outpass, -1, NULL, 0, iter, macmd);
  if (!rc) {
    BIO_printf(bio_err, "PKCS12_set_mac failed\n");
    goto err;
  }
  // Сохраняем контейнер в файле
  BIO_printf(bio_out, "Saving PKCS12 to '%s'\n", p12_path_new);
  bio_p12 = BIO_new_file(p12_path_new, "wb");
  i2d_PKCS12_bio(bio_p12, p12);
  BIO_free(bio_p12);
  PKCS12_free(p12);
  p12 = NULL;
  EVP_PKEY_free(pkey);
  pkey = NULL;

  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
  if (p12)
    PKCS12_free(p12);
    if (x)
        X509_free(x);
    if (pkey)
        EVP_PKEY_free(pkey);
```

```
    if (e)
        ENGINE_free(e);

    BIO_printf(bio_out, "lc_core_2012_pkcs12 test SUCCESS\n");
  if (bio_out)
    BIO_free(bio_out);
    return 0;
err:
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);
    if (x)
        X509_free(x);
    if (pkey)
        EVP_PKEY_free(pkey);
    if (e)
        ENGINE_free(e);
  ERR_print_errors(bio_err);
  if (bio_out)
    BIO_free(bio_out);
    return -1;
}
```

5.9 Особенности программирования для токенов

При программировании приложений LCSSL для токенов используются дополнительные управляющие команды при подключении ENGINE `lc_p11_core_2012`. В приведенном ниже примере организована универсальная загрузка ENGINE, в зависимости от аргументов главной функции main. Если аргументы не заданы, то загружается `lc_core_2012`, а если программа вызвана с аргументами `-e <engine_path>` и `-a <api_path>`, где `<engine_path>` – это путь к `lc_p11_core_2012`, а `<api_path>` – это путь к библиотеке PKCS#11, то будет загружен ENGINE `lc_p11_core_2012`.

```
int main(int argc, char **argv)
{
  ENGINE *e = NULL;
  int rc = 0;
  const char* param_28147 =
  "id-Gost28147-89-CryptoPro-A-ParamSet";
#ifdef WIN32
  static char *default_engine_path = "lc_core_2012";
#elif __APPLE__
```

```
  static char *default_engine_path =
   "/usr/local/lib/engines/liblc_core_2012.dylib";
#else
  static char *default_engine_path =
   "/usr/local/lib/engines/liblc_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;

  while((c = getopt(argc, argv, "e:a:"))!=-1){
      switch(c){
        case 'e':
      engine_path = BUF_strdup(optarg);
          break;
        case 'a':
      api_path = BUF_strdup(optarg);
          break;
      }
    }
  if (!engine_path) {
    engine_path = BUF_strdup(default_engine_path);
  }

    // Error messages output
    if (bio_err == NULL)
    if ((bio_err = BIO_new(BIO_s_file())) != NULL)
      BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    // Standard output
    if (bio_out == NULL)
        if ((bio_out=BIO_new(BIO_s_file())) != NULL)
          BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

    // Builtin engines should be loaded first
    // to access builtin "dynamic" engine then.
    ENGINE_load_builtin_engines();

  e = ENGINE_by_id("dynamic");
  if (e)
  {
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
```

```
    {
            BIO_printf(bio_err, "Loading ENGINE %s failed\n",
   engine_path);
            goto err;
    }
    if (api_path) {
      // В данном примере демонстрируется формирование engine_id
      // с явным указанием слота, если при вызове программы он
      // задан с флагом -s.
      char engine_id[128];
      if (slot) {
        sprintf(engine_id, "%s:%s", api_path, slot);
      } else {
        sprintf(engine_id, "%s", api_path);
      }
      if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", api_path, 0)
        ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", engine_id, 0)
        ||!ENGINE_ctrl_cmd_string(e, "OPEN_SESSION", "
   CKF_SERIAL_SESSION|CKF_RW_SESSION", 0)
        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "01234567", 0))
// Вариант с колбэком для ввода PIN: ... Verify, Prompt,
   Callback
//        ||!ENGINE_ctrl_cmd(e, "LOGIN", 1, "Enter PIN:",
   EVP_read_pw_string, 0))
// Этот вариант будет работать и в такой форме по умолчанию:
//        ||!ENGINE_ctrl_cmd(e, "LOGIN", 1, NULL, NULL, 0))
// или даже в такой (здесь OpenSSL не дает задавать NULL в треть
   ем аргументе,
// поэтому задается пустая строка, а Verify по умолчанию полагае
   тся равным 1):
//        ||!ENGINE_ctrl_cmd_string(e, "LOGIN", "", 0))
      {
        BIO_printf(bio_err, "ATTACH_API %s failed\n", api_path);
        goto err;
      }
    }
  }
  BIO_printf(bio_err, "Engine 0x%x loaded with id '%s'\n",
    e, ENGINE_get_id(e));


  rc = ENGINE_ctrl_cmd_string(e, "PARAMS_GOST28147",
            param_28147, 0);
    if (!rc) {
```

```
        BIO_printf(bio_err, "Setting PARAMS_GOST28147 failed\n")
    ;
        goto err;
    }
  if (engine_path)
    OPENSSL_free(engine_path);
  if (api_path)
    OPENSSL_free(api_path);

  // ...

  return rc;
}
```

В данном примере будет использован слот первого попавшегося токена, подключенного к библиотеке PKCS#11. Если нужно использовать слот с конкретным идентификатором, то этот идентификатор следует задать в команде ENGINE_ID после двоеточия, например:

```
if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", "etpkcs11g", 0)
  ||!ENGINE_ctrl_cmd_string(e, "ENGINE_ID", "etpkcs11g:2", 0)
// ...
```

При необходимости, можно загрузить несколько экземпляров одного и того же ENGINE и подключить их к разным токенам. В приведенных примерах идентификатор загруженного экземпляра ENGINE строится из имени библиотеки PKCS#11 и идентификатора слота. Вообще говоря, идентификатор экземпляра ENGINE может быть любым, лишь бы он был уникальным среди всех загруженных ENGINE.

Обратите внимание на закомментированные варианты выполнения логина с использованием колбэк-функции для ввода PIN токена. Здесь используется колбэк-функция OpenSSL `EVP_read_pw_string`. Прикладная программа может задать в управляющей команде LOGIN свою колбэк-функцию с таким же интерфейсом, как у `EVP_read_pw_string`:

```
int EVP_read_pw_string(char *buf,int length,const char *prompt,
   int verify);
```

При программировании приложений LCSSL, использующих токены, следует иметь в виду, что не каждый токен позволит импортировать закрытый ключ из файла и экспортировать значение закрытого ключа из объекта токена. Поэтому некоторые привычные функции OpenSSL могут не работать с токенами или работать по-другому.

При загрузке закрытого ключа из файла в памяти приложения создается объект `EVP_PKEY` с реальным значением закрытого ключа. Для сохранения закрытого ключа на токен следует воспользоваться функцией `STORE_store_private_key`. После этого

следует уничтожить интерфейсный объект `EVP_PKEY`, чтобы в памяти не осталось значение закрытого ключа. Например:

```
OPENSSL_ITEM attr_keypair[] = {
    {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
    {STORE_ATTR_END, NULL, 0, NULL}
};
EVP_PKEY *pkey = NULL;
char *pkeyfile = "pkey.pem";
BIO *pkeybio = NULL;
char *label = "store_pkey";
STORE *st = NULL;

// ...

st = STORE_new_engine(e);
if (!st) {
  BIO_printf(bio_err, "ENGINE has no STORE interface\n");
  goto end;
}
attr_keypair[0].value = label;
attr_keypair[0].value_size = strlen(label)+1;
pkeybio = BIO_new_file(pkeyfile, "rb");
if (!pkeybio) {
  BIO_printf(bio_err, "BIO_new_file failed\n");
  goto end;
}
// Данная функция закружает закрытый ключ из файла в память прил
   ожения.
pkey = PEM_read_bio_PrivateKey(pkeybio, &pkey, NULL, NULL);
if (!pkey) {
  BIO_printf(bio_err, "PEM_read_bio_PrivateKey failed\n");
  goto end;
}
// Данная функция сохраняет объект закрытого ключа на токене.
if ( 0 >= STORE_store_private_key( st, pkey, attr_keypair, NULL
   ) )
{
  BIO_printf(bio_err, "STORE_store_private_key() failed\n" );
  goto end;
}
BIO_printf(bio_err,
  "Private key stored with label '%s'\n",
  label);
```

```
// ...
end:
if (pkey) EVP_PKEY_free(pkey);
if (pkeybio) BIO_free(pkeybio);
if (st) STORE_free(st);
// ...
```

Если значение закрытого ключа хранится в объекте токена, то прикладной программе оно может быть не доступно. Поэтому для получения интерфейса `EVP_PKEY` к закрытому ключу лучше всего воспользоваться объектом соответствующего сертификата X509:

```
X509 *x = ...
EVP_PKEY *pkey = X509_get_pubkey(x);
...
```

Полученный таким образом интерфейсный объект pkey при использовании ENGINE `lc_p11_core_2012` содержит идентификатор закрытого ключа, по которому ENGINE найдет закрытый ключ на токене и будет производить операции с ним, вызывая функции PKCS#11.

Заметим, что такой способ доступа к закрытому ключу на токене через сертификат будет работать, только если в атрибуте `CKA_ID` объекта закрытого ключа содержится значение функции SHA-1 от значения открытого ключа в DER-кодировке OCTET STRING. В продуктах "ЛИССИ-Софт"это делается при создании ключевой пары.

Кроме интерфейса STORE, у `lc_p11_core_2012` имеется реализация еще двух функций для доступа к объектам ключевой пары:

```
EVP_PKEY *ENGINE_load_private_key(ENGINE *e, const char *key_id,
  UI_METHOD *ui_method, void *callback_data);
EVP_PKEY *ENGINE_load_public_key(ENGINE *e, const char *key_id,
  UI_METHOD *ui_method, void *callback_data);
```

Интерпретация параметра `key_id` зависит от реализации. В общем смысле подразумевается, что это символьная строка, как-то идентифицирующая соответствующий ключ. В приложениях OpenSSL этот параметр обычно является именем файла, содержащего ключ. В нашем случае ключ находится на токене, поэтому применяется другая интерпретация, позволяющая получать интерфейс закрытого ключа либо по имени файла соответствующего сертификата, либо по метке ключа на токене.

В качестве `key_id` при вызове этих функций передается либо путь к PEM-файлу сертификата закрытого ключа, либо метка закрытого ключа на токене. Сначала проверяется путь к файлу сертификата, а если не получилось его загрузить, то производится поиск ключа на токене по метке. Параметры `ui_method` и `callback_data` в текущей реализации полагаются равными NULL.

Данный способ интерпретации параметра `key_id` в значительной степени обусловлен потребностями утилит командной строки lcssl и store, в аргументах которых

должна задаваться только символьная информация.

Иногда целесообразно начальные операции с библиотекой PKCS#11 выполнить через ее родной интерфейс, после чего передать уже готовые список функций и открытую на нужном слоте сессию прямо в контекст ENGINE `lc_p11_core_2012`:

```
ENGINE *e = NULL;
#ifdef WIN32
  static char *engine_path = "lc_p11_core_2012";
#elif __APPLE__
  static char *engine_path =
    "/usr/local/lib/engines/liblc_p11_core_2012.dylib";
#else
  static char *engine_path =
    "/usr/local/lib/engines/liblc_p11_core_2012.so";
#endif
CK_FUNCTION_LIST_PTR funcs = NULL;
CK_INVALID_HANDLE sess = CK_INVALID_HANDLE;

e = ENGINE_by_id("dynamic");
if (e)
{
  if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
    || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
  {
        BIO_printf(bio_err,
          "Loading ENGINE %s failed\n", engine_path);
        goto err;
  }
}
// Далее загружается библиотека PKCS#11,
// получается адрес списка функций funcs,
// функциями PKCS#11 инициализируется библиотека,
// получается список слотов, определяется нужный слот,
// открывается сессия с хэндлом sess и выполняется логин.
// ...

// А теперь готовые funcs и sess передаются в контекст
// ранее загруженного ENGINE lc_p11_core_2012:
if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", funcs, 0)
  ||!ENGINE_ctrl_cmd_string(e, "SET_SESSION", &sess, 0))
{
  BIO_printf(bio_err, "ATTACH_API with SET_SESSION failed\n");
  goto err;
}
```

```
// Теперь ENGINE подключен к нужному токену и готов
// к работе с ним.
// ...
```

Используя данный способ, можно загрузить только один экземпляр ENGINE, а затем динамически подключать его к разным токенам и даже к токенам разных библиотек PKCS#11, не закрывая открытые на соответствующих слотах сессии. При этом, важно понимать, что в данный момент времени экземпляр ENGINE работает только с одним токеном просто потому, что в контексте ENGINE сохраняется хэндл одной-единственной сессии на указанном слоте соответствующей библиотеки.

Более подробно можно ознакомиться с использованием данного способа в тестовом примере `p11_api_example`:

Листинг 5.12: p11_api_example.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <openssl/bio.h>
#include <openssl/crypto.h>
#include <openssl/x509.h>
#include <openssl/pem.h>
#include <openssl/engine.h>
#include <openssl/store.h>
#include <openssl/err.h>
#ifdef WIN32
#include <windows.h>
#else
#include <unistd.h>
#include <dlfcn.h>
#endif
#include "pkcs11types.h"
#include "pkcs11ctrl.h"
#include "lc_keypair_2012.h"
#include "lc_defs_2012.h"
#include "getopt.h"

void bio_print_hex(BIO *bio, unsigned char *buf, size_t len) {
    size_t i;
    for (i=0; i<len; i++)
        BIO_printf(bio, "%02X%c", buf[i],
        ((i+1)==len||!((i+1)%16))?'\n':' ');
}

BIO *bio_err = NULL;
```

```
BIO *bio_out = NULL;

void print_info(CK_INFO *ainfo);
void print_slot_info(CK_SLOT_INFO *sinfo);
void print_token_info(CK_TOKEN_INFO *tinfo);

void print_info(CK_INFO *ainfo)
{
    printf("API Info:\n");
    printf("Cryptoki Version: {%d, %d}\n",
        ainfo->cryptokiVersion.major, ainfo->cryptokiVersion.
   minor);
    printf("Manufacturer ID: '%.32s'\n", ainfo->manufacturerID);
    printf("Flags: 0x%x\n", ainfo->flags);
    printf("Library Description: '%.32s'\n", ainfo->
   libraryDescription);
    printf("Library Version: {%d, %d}\n",
        ainfo->libraryVersion.major, ainfo->libraryVersion.minor
   );
}

void print_slot_info(CK_SLOT_INFO *sinfo)
{
    printf("Slot Info:\n");
    printf("Slot Description: '%.64s'\n", sinfo->slotDescription
   );
    printf("Manufacturer ID: '%.32s'\n", sinfo->manufacturerID);
    printf("Flags: 0x%x\n", sinfo->flags);
    printf("Hardware Version: {%d, %d}\n",
        sinfo->hardwareVersion.major, sinfo->hardwareVersion.
   minor);
    printf("Firmware Version: {%d, %d}\n",
        sinfo->firmwareVersion.major, sinfo->firmwareVersion.
   minor);
}

void print_token_info(CK_TOKEN_INFO *tinfo)
{
    printf("Token Info:\n");
    printf("\tLabel: '%.32s'\n", tinfo->label);
    printf("\tManufacturer: '%.32s'\n", tinfo->manufacturerID);
    printf("\tModel: '%.16s'\n", tinfo->model);
    printf("\tSerial Number: '%.16s'\n", tinfo->serialNumber);
    printf("\tFlags: 0x%X\n", tinfo->flags);
```

```
    printf("\tSessions: %d/%d\n", tinfo->ulSessionCount,
        tinfo->ulMaxSessionCount);
    printf("\tR/W Sessions: %d/%d\n",
        tinfo->ulRwSessionCount, tinfo->ulMaxRwSessionCount);
    printf("\tPIN Length: %d-%d\n", tinfo->ulMinPinLen,
        tinfo->ulMaxPinLen);
    printf("\tPublic Memory: 0x%X/0x%X\n",
        tinfo->ulFreePublicMemory, tinfo->ulTotalPublicMemory);
    printf("\tPrivate Memory: 0x%X/0x%X\n",
        tinfo->ulFreePrivateMemory, tinfo->ulTotalPrivateMemory)
   ;
    printf("\tHardware Version: %d.%d\n", tinfo->hardwareVersion
   .major,
        tinfo->hardwareVersion.minor);
    printf("\tFirmware Version: %d.%d\n",
        tinfo->firmwareVersion.major,
        tinfo->firmwareVersion.minor);
    printf("\tTime: %.16s\n", tinfo->utcTime);
}

int main(int argc, char **argv)
{
#ifdef WIN32
    char *default_api_path = "ls11sw2012";
    HMODULE dll = NULL;
#elif __APPLE__
    char *default_api_path = "libls11sw2012.dylib";
  void *dll = NULL;
#else
    char *default_api_path = "libls11sw2012.so";
  void *dll = NULL;
#endif
    CK_RV rc = CKR_OK;
    CK_C_GetFunctionList pcGetFunctionList = NULL;  // pointer
   to C_GetFunctionList
    CK_FUNCTION_LIST_PTR funcs = NULL;          // pointer to
   PKCS #11 function list
    CK_ULONG ulSlotCount = 0;
    CK_SLOT_ID_PTR pSlotList = NULL;      // pointer to slot
   list
    CK_SLOT_ID SlotId = (CK_SLOT_ID)-1;
    CK_INFO ainfo;
    CK_SLOT_INFO sinfo;
    CK_TOKEN_INFO tinfo;
```

```
    char *user_pin = "01234567";
    CK_FLAGS sess_flags = CKF_SERIAL_SESSION | CKF_RW_SESSION;
    CK_SESSION_HANDLE hSession = CK_INVALID_HANDLE;
    static CK_BBOOL bTrue       = CK_TRUE,
        bFalse      = CK_FALSE;
    static CK_OBJECT_CLASS ulClass_PrivKey = CKO_PRIVATE_KEY;
    static CK_OBJECT_CLASS ulClass_PubKey = CKO_PUBLIC_KEY;
  ENGINE *e = NULL;
  STORE *st = NULL;
  EVP_PKEY *pkey = NULL;
    EVP_PKEY_CTX *ctx = NULL;
  unsigned char msg[] = {
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x00,
  };
    EVP_MD *md = NULL;
    EVP_MD_CTX *md_ctx = NULL;
    unsigned char hash[64];
  unsigned int hash_len = sizeof(hash);
    unsigned char sig[128];
    size_t sig_len = sizeof(sig);
  LC_GOST2012_KEYPAIR *lcc_gost2012_key = NULL;
#ifdef WIN32
  static char *default_engine_path = "lc_p11_core_2012";
#elif __APPLE__
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_p11_core_2012.dylib";
#else
  static char *default_engine_path = "/usr/local/lib/engines/
   liblc_p11_core_2012.so";
#endif
  char *engine_path = NULL;
  char *api_path = NULL;
    extern char *optarg;
    int c;
  BIO *bio_pkey = NULL;
  char *pkey_pem_path = "test_cp_key.pem";
  char *cert_pem_path = "test_cp_cert.pem";
  char label[] = "test_cp";
  OPENSSL_ITEM attr_label[] = {
        {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
        {STORE_ATTR_END, NULL, 0, NULL}
  };
  OPENSSL_ITEM attr_id[] = {
```

```
        {STORE_ATTR_KEYID, NULL, 0, NULL},
        {STORE_ATTR_END, NULL, 0, NULL}
};


while((c = getopt(argc, argv, "a:s:"))!=-1){
    switch(c){
      case 'a':
    api_path = BUF_strdup(optarg);
        break;
      case 's':
    SlotId = atoi(BUF_strdup(optarg));
        break;
    }
  }
if (!engine_path) {
  engine_path = BUF_strdup(default_engine_path);
}
if (!api_path) {
  api_path = BUF_strdup(default_api_path);
}

  // Error messages output
  if (bio_err == NULL)
  if ((bio_err = BIO_new(BIO_s_file())) != NULL)
    BIO_set_fp(bio_err, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

  // Standard output
  if (bio_out == NULL)
      if ((bio_out=BIO_new(BIO_s_file())) != NULL)
        BIO_set_fp(bio_out, stdout, BIO_NOCLOSE|BIO_FP_TEXT);

BIO_printf(bio_err, "%s PKCS#11 API & OpenSSL test\n",
 engine_path);
  // Builtin engines should be loaded first
  // to access builtin "dynamic" engine then.
  ENGINE_load_builtin_engines();

// Загружаем ENGINE lc_p11_core_2012.
// По умолчанию, он загружается с ID "ls11sw2012",
// но мы вообще не будем использовать здесь его ID.
e = ENGINE_by_id("dynamic");
if (e)
{
```

```
    if (!ENGINE_ctrl_cmd_string(e, "SO_PATH", engine_path, 0)
      || !ENGINE_ctrl_cmd_string(e, "LOAD", NULL, 0))
    {
      BIO_printf(bio_err,
            "Loading ENGINE %s failed\n", engine_path);
      goto err;
    }
  }

    // Шаги PKCS#11:
    // Загружаем библиотеку PKCS#11.
#ifdef WIN32
    // Extention ".dll" will be added automatically to api_path
    dll = LoadLibrary(api_path);
#else
    // Prefix "lib" and extention ".so" or ".dylib" should be
   present in api_path
    dll = dlopen(api_path, RTLD_NOW);
#endif
    if (dll == NULL) {
        fprintf(stderr, "LoadLibrary failed\n");
        goto err;
    }
    // Получаем список функций PKCS#11.
#ifdef WIN32
    pcGetFunctionList = (CK_C_GetFunctionList)GetProcAddress(dll
   , "C_GetFunctionList");
#else
    pcGetFunctionList = (CK_C_GetFunctionList)dlsym(dll, "
   C_GetFunctionList");
#endif
    if (pcGetFunctionList == NULL) {
        fprintf(stderr, "Can't get C_GetFunctionList address\n")
   ;
        goto err;
    }
    rc = pcGetFunctionList(&funcs);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_GetFunctionList failed, rc = 0x%x\n",
    rc);
        goto err;
    }
    // Инициализируем PKCS#11 API
    rc = funcs->C_Initialize(NULL);
```

```
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Initialize failed: 0x%x\n", rc);
        goto err;
    }
    // Получаем информацию о библиотеке.
    rc = funcs->C_GetInfo(&ainfo);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_GetInfo failed: 0x%x\n", rc);
        goto err;
    }
    print_info(&ainfo);

    // Получаем список слотов с токенами.
    rc = funcs->C_GetSlotList(CK_TRUE, NULL, &ulSlotCount);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_GetSlotList failed, rc = 0x%x\n", rc)
;
        goto err;
    }
    if (ulSlotCount > 0)
    {
        pSlotList = (CK_SLOT_ID_PTR) malloc(ulSlotCount*sizeof(
CK_SLOT_ID));
        rc = funcs->C_GetSlotList(CK_TRUE, pSlotList, &
ulSlotCount);
        if (rc != CKR_OK) {
            fprintf(stderr, "C_GetSlotList failed, rc = 0x%x\n",
 rc);
            goto err;
        }
    } else {
        fprintf(stderr, "No slots with token present found\n",
rc);
        rc = CKR_FUNCTION_FAILED;
        goto err;
    }
  if (SlotId == -1) {
    // Используем первый попавшийся слот с токеном.
    SlotId = pSlotList[0];
  }
    // Получаем информацию о слоте.
    rc = funcs->C_GetSlotInfo(SlotId, &sinfo);
    if (rc != CKR_OK) {
        goto err;
```

```
    }
    print_slot_info(&sinfo);
    // Получаем информацию о токене.
    rc = funcs->C_GetTokenInfo(SlotId, &tinfo);
    if (rc != CKR_OK) {
        goto err;
    }
    print_token_info(&tinfo);
  // Открываем сессию на слоте.
    rc = funcs->C_OpenSession(SlotId,
        sess_flags,
        NULL,
        NULL,
        &hSession);
    if (rc != CKR_OK) {
        goto err;
    }
  // Выполняем логин.
    rc = funcs->C_Login(hSession, CKU_USER, (CK_UTF8CHAR_PTR)
   user_pin, strlen(user_pin));
    if (rc != CKR_OK) {
    fprintf(stderr, "C_Login failed, rc = 0x%x\n", rc);
        goto err;
    }

  // Шаги OpenSSL:
  // Готовые funcs и sess передаются в контекст
  // ранее загруженного ENGINE lc_p11_core_2012:
  if (!ENGINE_ctrl_cmd_string(e, "ATTACH_API", (const char *)
   funcs, 0)
    ||!ENGINE_ctrl_cmd_string(e, "SET_SESSION", (const char *)&
   hSession, 0))
  {
    BIO_printf(bio_err, "ATTACH_API with SET_SESSION failed\n");
    rc = (CK_RV)-1;
    goto err;
  }
  // Теперь ENGINE подключен к нужному токену и готов
  // к работе с ним.

  // Получаем из ENGINE интерфейс STORE.
  st = STORE_new_engine(e);
  if (!st) {
    BIO_printf(bio_err, "ENGINE has no STORE interface\n");
```

```
        rc = (CK_RV)-1;
    goto err;
}

// Закружаем закрытый ключ из незашифрованного PEM-файла
bio_pkey = BIO_new_file(pkey_pem_path, "rb");
pkey = PEM_read_bio_PrivateKey(bio_pkey, NULL, NULL, NULL);
BIO_free(bio_pkey);
if (!pkey) {
        BIO_printf(bio_err, "Can't read private key from PEM
 file\n");
        rc = (CK_RV)-1;
      goto err;
}
lcc_gost2012_key = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(pkey);
if (lcc_gost2012_key->is_real_priv_key) {
    BIO_printf(bio_out, "Imported private key value:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_value,
      lcc_gost2012_key->priv_key_len);
}
BIO_printf(bio_out, "Imported private key ID:\n");
bio_print_hex(bio_out, lcc_gost2012_key->priv_key_id,
    sizeof(lcc_gost2012_key->priv_key_id));

// Выгружаем закрытый ключ на токен через интерфейс STORE.
attr_label[0].value = label;
attr_label[0].value_size = strlen(label)+1;
if ( 0 >= STORE_store_private_key( st, pkey, attr_label, NULL
 ) )
{
    BIO_printf(bio_err, "STORE_store_private_key failed\n" );
        rc = (CK_RV)-1;
    goto err;
}
EVP_PKEY_free(pkey);
pkey = NULL;

// Получаем интерфейс закрытого ключа на токене
// по имени файла сертификата.
pkey = ENGINE_load_private_key(e, cert_pem_path, NULL, NULL);
if (!pkey) {
    BIO_printf(bio_err, "ENGINE_load_private_key failed\n" );
        rc = (CK_RV)-1;
    goto err;
```

```
  }
  BIO_printf(bio_out,
    "Private key ID loaded by certificate file path %s:\n",
    cert_pem_path);
  bio_print_hex(bio_out, lcc_gost2012_key->priv_key_id,
    sizeof(lcc_gost2012_key->priv_key_id));
  EVP_PKEY_free(pkey);
  pkey = NULL;

  // Другой способ:
  // Получаем интерфейс закрытого ключа на токене
  // по его метке.
  pkey = ENGINE_load_private_key(e, label, NULL, NULL);
  if (!pkey) {
    BIO_printf(bio_err, "ENGINE_load_private_key failed\n");
        rc = (CK_RV)-1;
    goto err;
  }
  BIO_printf(bio_out,
    "Private key ID loaded by label '%s':\n", label);
  bio_print_hex(bio_out, lcc_gost2012_key->priv_key_id,
    sizeof(lcc_gost2012_key->priv_key_id));
  EVP_PKEY_free(pkey);
  pkey = NULL;

  // Удаляем закрытый ключ на токене по его ID
  // с помощью интерфейса STORE.
  attr_id[0].value = lcc_gost2012_key->priv_key_id;
  attr_id[0].value_size = sizeof(lcc_gost2012_key->priv_key_id);
  rc = STORE_delete_private_key(st, attr_id, NULL);
  if (rc != 1) {
    BIO_printf(bio_err, "STORE_delete_private_key failed\n");
        rc = (CK_RV)-1;
    goto err;
  }

  // Проверяем работу различных прикладных алгоритмов.
  // Дайджест 3411-94.
    BIO_printf(bio_out, "Make NID_id_GostR3411_94 message digest
   \n");
    md = (EVP_MD *)ENGINE_get_digest(e, NID_id_GostR3411_94);
    if(!md) {
        BIO_printf(bio_err, "Can't get digest
   NID_id_GostR3411_94\n");
```

```
        rc = (CK_RV)-1;
        goto err;
    }
    md_ctx = EVP_MD_CTX_create();
    rc = EVP_DigestInit_ex(md_ctx, md, e);
    if (rc != 1) {
        rc = (CK_RV)-1;
        goto err;
    }
    rc = EVP_DigestUpdate(md_ctx, msg, sizeof(msg));
    if (rc != 1) {
        rc = (CK_RV)-1;
        goto err;
    }
    rc = EVP_DigestFinal(md_ctx, hash, &hash_len);
    if (rc != 1) {
        rc = (CK_RV)-1;
        goto err;
    }
    EVP_MD_CTX_destroy(md_ctx);

  // Генерируем ключевую пару
    BIO_printf(bio_out,
    "Generate NID_id_GostR3410_2001 key pair with %s\n",
    SN_id_GostR3410_2001_CryptoPro_A_ParamSet);
  {
    OPENSSL_ITEM attr_keypair[] = {
            {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
            {STORE_ATTR_END, NULL, 0, NULL}
    };
    OPENSSL_ITEM param_keypair[] = {
            {STORE_PARAM_EVP_TYPE, NULL, 0, NULL},
            {STORE_PARAM_KEY_PARAMETERS, NULL, 0, NULL},
            {0, NULL, 0, NULL}
    };
    int pkey_nid = NID_id_GostR3410_2001;
    int parnid = NID_id_GostR3410_2001_CryptoPro_A_ParamSet;

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    param_keypair[0].value = &pkey_nid;
    param_keypair[0].value_size = sizeof(pkey_nid);
    param_keypair[1].value = &parnid;
    param_keypair[1].value_size = sizeof(parnid);
```

```
    pkey = STORE_generate_key(st, attr_keypair, param_keypair);
    if (!pkey)
    {
      BIO_puts(bio_err, "Error generating key\n");
      ERR_print_errors(bio_err);
      rc = (CK_RV)-1;
      goto err;
    }
    BIO_printf(bio_err,
      "Keypair '%s' with algorithm %s and paramset %s generated
  in store\n",
      label, OBJ_nid2sn(pkey_nid), OBJ_nid2sn(parnid));
  }
  lcc_gost2012_key = (LC_GOST2012_KEYPAIR *)EVP_PKEY_get0(pkey);
  if (lcc_gost2012_key->is_real_priv_key) {
    BIO_printf(bio_out, "Generated private key:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_value,
      lcc_gost2012_key->priv_key_len);
  } else {
    BIO_printf(bio_out, "Generated private key ID:\n");
    bio_print_hex(bio_out, lcc_gost2012_key->priv_key_id,
      sizeof(lcc_gost2012_key->priv_key_id));
  }
    // Генерация ЭЦП
    memset(sig, 0, sizeof(sig));
    // Создаем контекст ЭЦП для закрытого ключа с помощью ENGINE
    BIO_printf(bio_out, "Create sign context\n");
    ctx = EVP_PKEY_CTX_new(pkey, e);
    if (ctx) {
        // Подписываем дайджест hash.
        BIO_printf(bio_out, "Sign message hash\n");
        rc = EVP_PKEY_sign_init(ctx);
        if (rc == 1) {
            rc = EVP_PKEY_sign(ctx, sig, &sig_len, hash,
   hash_len);
            if (!rc) {
                rc = (CK_RV)-1;
                goto err;
            }
            BIO_printf(bio_out, "Signature generated OK\n");
        } else {
            rc = (CK_RV)-1;
            goto err;
        }
```

```
        EVP_PKEY_CTX_free(ctx);
    }

  // Проверка ЭЦП.
    // Создаем контекст ЭЦП для открытого ключа с помощью ENGINE
   .
    BIO_printf(bio_out, "Create verify context\n");
    ctx = EVP_PKEY_CTX_new(pkey, e);
    if (ctx) {
        // Проверяем ЭЦП
        BIO_printf(bio_out, "Verify message hash signature\n");
        rc = EVP_PKEY_verify_init(ctx);
        if (rc == 1) {
            rc = EVP_PKEY_verify(ctx, sig, sig_len, hash,
  hash_len);
            if (!rc) {
                rc = (CK_RV)-1;
                goto err;
            }
            BIO_printf(bio_out, "Signature verified OK\n");
        } else {
            rc = (CK_RV)-1;
            goto err;
        }
        EVP_PKEY_CTX_free(ctx);
    }
    EVP_PKEY_free(pkey);
  pkey = NULL;

  if (st) {
    // Remove created token objects
    OPENSSL_ITEM attr_keypair[] = {
          {STORE_ATTR_FRIENDLYNAME, NULL, 0, NULL},
          {STORE_ATTR_END, NULL, 0, NULL}
    };

    attr_keypair[0].value = label;
    attr_keypair[0].value_size = strlen(label)+1;
    rc = STORE_delete_private_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_private_key failed\n" );
      rc = (CK_RV)-1;
      goto err;
    }
```

```
    rc = STORE_delete_public_key(st, attr_keypair, NULL);
    if (rc != 1) {
      BIO_printf(bio_err, "STORE_delete_public_key failed\n");
      rc = (CK_RV)-1;
      goto err;
    }
  }

  // Логаут
    rc = funcs->C_Logout(hSession);
    if (rc != CKR_OK) {
        goto err;
    }
  // Закрываем сессию
    rc = funcs->C_CloseSession(hSession);
    if (rc != CKR_OK) {
        goto err;
    }

    BIO_printf(bio_err, "Free OpenSSL interface objects\n");

  // Освобождаем ENGINE
  if (e)
    ENGINE_free(e);
  e = NULL;

  if (bio_out)
    BIO_free(bio_out);

  BIO_printf(bio_err, "%s PKCS#11 API & OpenSSL test SUCCESS\n",
    engine_path);

    fprintf(stdout, "SUCCESS\n");
    rc = 0;
    return rc;

err:
    return rc;
}
```

6 Утилиты командной строки

Использование утилит командной строки lcssl и store предполагает, что LCSSL и ENGINE инсталлированы в системе штатным образом.

6.1 Файл конфигурации

Оригинальный OpenSSL 1.0.0 предоставляет возможность выполнения различных операций с помощью утилиты lcssl. Для использования с `lc_core_2012` или `lc_p11_core_2012` утилиты lcssl нужно предварительно в файле конфигурации openssl.cnf разместить следующие секции:

6.1.1 Windows

```
openssl_conf = openssl_def

[openssl_def]
engines = engine_section

[engine_section]
lcc2012 = lc_core_2012_section
#lccp112012 = lc_p11_core_2012_section

[lc_p11_core_2012_section]
dynamic_path = lc_p11_core_2012
default_algorithms = ALL
ATTACH_API = ls11sw2012
ENGINE_ID = ls11sw2012
OPEN_SESSION = CKF_SERIAL_SESSION|CKF_RW_SESSION
LOGIN = 01234567
PARAMS_GOST28147 = id-Gost28147-89-CryptoPro-A-ParamSet

[lc_core_2012_section]
dynamic_path = lc_core_2012
default_algorithms = ALL
PARAMS_GOST28147 = id-Gost28147-89-CryptoPro-A-ParamSet
```

6.1.2 Linux

```
openssl_conf = openssl_def

[openssl_def]
engines = engine_section

[engine_section]
lcc2012 = lc_core_2012_section
#lccp112012 = lc_p11_core_2012_section

[lc_core_2012_section]
dynamic_path = /usr/local/lib/engines/liblc_core_2012.so
default_algorithms = ALL
PARAMS_GOST28147 = id-Gost28147-89-CryptoPro-A-ParamSet

[lc_p11_core_2012_section]
dynamic_path = /usr/local/lib/engines/liblc_p11_core_2012.so
default_algorithms = ALL
ATTACH_API = /usr/local/lib/libls11sw2012.so
ENGINE_ID = ls11sw2012
OPEN_SESSION = CKF_SERIAL_SESSION|CKF_RW_SESSION
LOGIN = 01234567
PARAMS_GOST28147 = id-Gost28147-89-CryptoPro-A-ParamSet
```

В такой конфигурации утилита lcssl будет использовать ENGINE `lc_core_2012`. Для использования ENGINE `lc_p11_core_2012` с библиотекой PKCS#11 ls11sw2012 следует закомментировать строку lcc2012 в `engine_section` и раскомментировать строку lccp112012. Для использования `lc_p11_core_2012` с другой библиотекой PKCS#11 нужно добавить аналогичную секцию, заменив имя библиотеки.

Заметим, что ТК 26 рекомендует использовать для симметричного шифрования новый набор параметров `id-tc26-gost-28147-89-param-A` (1.2.643.7.1.2.5.1.1), который поддерживается библиотекой lcc2012 и ENGINE `lc_core_2012`.

По умолчанию, путь к файлу конфигурации в утилите определяется значением переменной среды `LCSSL_CONF`.

В некоторых вариантах использования утилиты можно явно указать путь к файлу конфигурации с помощью опции -config. Это предпочтительно в тех случаях, когда желательно исключить путаницу при использовании различных ENGINE, предоставляющих аналогичные интерфейсы.

По умолчанию, путь к динамическим библиотекам ENGINE в утилите определяется значением переменной среды `LCSSL_ENGINES`.

Заметим, что для работы описанных выше примеров программ файл конфигурации не нужен, потому что все конфигурационные действия производится в них непосредственно с помощью управляющих команд.

6.2 Утилита lcssl

6.2.1 Генерация дайджеста

По алгоритму ГОСТ Р34.11-94:

```
>lcssl dgst -binary -md_gost94 -out digest_94.bin data.bin
```

По алгоритму ГОСТ Р34.11-2012-256:

```
>lcssl dgst -binary -gost3411-2012-256 -out digest-2012-256.bin data.bin
```

По алгоритму ГОСТ Р34.11-2012-512:

```
>lcssl dgst -binary -gost3411-2012-512 -out digest-2012-512.bin data.bin
```

6.2.2 Генерация HMAC

```
>lcssl dgst -binary -md_gost94
-hmac
0102030405060708091011121314151617181920212223242526272829303132
-out hmac.bin data.bin
```

Заменяя в этой команде `md_gost94` на gost3411-2012-256 или gost3411-2012-512, можно получить HMAC по другим алгоритмам.

6.2.3 Генерация имитовставки

```
>lcssl dgst -mac gost-mac -binary -macopt
hexkey:
3132333435363738393031323334353637383930313233343536373839303132
-out mac.bin data.bin
```

6.2.4 Симметричное шифрование

Зашифрование

```
>lcssl enc -e -gost89 -out ciphertext.bin -in plaintext.bin
-K 0102030405060708091011121314151617181920212223242526272829303132
-iv 0102030405060708
```

Расшифрование

```
>lcssl enc -d -gost89 -out decryptedtext.bin -in ciphertext.bin
-K 0102030405060708091011121314151617181920212223242526272829303132
-iv 0102030405060708
```

6.2.5 Генерация ключевой пары

Генерация ключевой пары утилитой genpkey

Для ГОСТ Р34.10-2001:

```
>lcssl genpkey -algorithm gost2001 -pkeyopt paramset:A
-out keypair.pem
```

Для ГОСТ Р34.10-2012-256:

```
>lcssl genpkey -algorithm gost3410-2012-256 -pkeyopt paramset:A
-out keypair.pem
```

Для ГОСТ Р34.10-2012-512:

```
>lcssl genpkey -algorithm gost3410-2012-512 -pkeyopt paramset:A
-out keypair.pem
```

При использовании утилиты genpkey генерируется новая ключевая пара и затем закрытый ключ сохраняется в файле keypair.pem.

6.2.6 Генерация открытого ключа

Получить файл открытого ключа по закрытому ключу можно следующей командой:

```
>lcssl pkey -in keypair.pem -pubout -out pubkey.pem
```

6.2.7 Генерация и проверка ЭЦП

Генерация ЭЦП

```
>lcssl pkeyutl -sign -in digest.bin -out sig.bin
-inkey keypair.pem
```

Проверка ЭЦП

```
>lcssl pkeyutl -verify -in digest.bin -sigfile sig.bin -pubin
-inkey pubkey.pem
```

6.2.8 Экспорт в контейнер PKCS12

При создании выходного контейнера будет запрошен пароль. В данном примере зкспортируемые закрытый ключ и сертификат находятся в одном PEM-файле. Если они находятся в разных файлах, то вместо –in нужно использовать –inkey и –certfile соответственно.

```
>lcssl pkcs12 -export -in test_cp.pem -name test_cp
-keypbe gost89 -certpbe gost89 -macalg md_gost94
-out test_cp.p12
```

6.2.9 Импорт из контейнера PKCS12

Перед импортом из входного контейнера будет запрошен пароль. Закрытый ключ и сертификат будут выгружены в один PEM-файл.

```
>lcssl pkcs12 -in test_cp.p12 -nodes -out test_cp.pem
```

Если нужно выгрузить закрытый ключ и сертификат в разные файлы, то это делается в два приема:

```
>lcssl pkcs12 -in test_cp.p12 -nodes -nocerts -out test_cp_key.pem
>lcssl pkcs12 -in test_cp.p12 -nodes -nokeys -out test_cp_cert.pem
```

6.2.10 Защищенное соединение по HTTPS

Следующая команда устанавливает анонимное соединение по протоколу HTTPS с сайтом КриптоПро:

```
>lcssl s_client -connect cryptopro.ru:443 -CAfile cp-ca.cer
```

В файле cp-ca.cer должен храниться доверенный сертификат УЦ Крипто-Про. После установления соединения должна быть получена примерно следующая информация на экране:

```
...
---
No client certificate CA names sent
---
SSL handshake has read 1180 bytes and written 422 bytes
---
New, TLSv1/SSLv3, Cipher is GOST2001-GOST89-GOST89
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol  : TLSv1
    Cipher    : GOST2001-GOST89-GOST89
    Session-ID: C8E75F3953B88F5AEED116B16F78CA4CB
94FEA26160D6CD95952D020957968AF
    Session-ID-ctx:
    Master-Key: 745FB722F451CEB7180664DEA1F196A6
DFBF89642811D8C70E067598BB8496E2
38DC4D5407C6F96A3BC2D3D06839E8B7
    Key-Arg   : None
    PSK identity: None
```

```
    PSK identity hint: None
    Start Time: 1295435107
    Timeout   : 300 (sec)
    Verify return code: 0 (ok)
---
```

Теперь защищенное соединение установлено. Далее можно, к примеру, ввести команду для скачивания корневой страницы сайта. Для этого нужно ввести с клавиатуры следующую команду и нажать ENTER:

```
>GET / HTTP/1.0
```

Затем нужно еще раз нажать клавишу ENTER для завершения сеанса. На экране должен появиться исходный HTML-код корневой страницы сайта, после чего сеанс завершится.

Для установки защищенного авторизованного соединения нужно указать в команде пользовательский закрытый ключ и сертификат. В данном примере они содержатся в одном файле `test_cp.pem`. Заметим, что используется другой порт – 4444.

```
>lcssl s_client -connect cryptopro.ru:4444 -CAfile cp-ca.cer -cert test_cp.pem
```

После установки соединения с сайтом Крипто-Про тестовую страницу с авторизованным доступом можно прочитать командой:

```
>GET /test/tls-cli.asp HTTP/1.0
```

6.2.11 Создание тестового УЦ

Для проверки работы команд УЦ можно организовать простейший тестовый УЦ. Для этого достаточно создать папку ./demoCA, а в ней папки private, newcerts и два файла: пустой файл index.txt и файл serial, содержащий 01. После этого, следующей командой req нужно создать ключевую пару УЦ и самоподписанный сертификат:

```
>lcssl req -x509 -newkey gost2001 -pkeyopt paramset:XA
-keyout ./demoCA/private/cakey.pem -out ./demoCA/cacert.pem
```

Заметим, что если ключевая пара создается на токене, то в файле cakey.pem содержится не значение закрытого ключа, а его идентификатор. Использование закрытого ключа по идентификатору обеспечивается соответствующим ENGINE.

Для того, чтобы тестовый УЦ не зависел от токена, можно создать его и использовать с ENGINE `lc_core_2012`. Для этого нужно не забыть прописать соответствующую секцию в файле конфигурации:

```
[engine_section]
lcc2012 = lc_core_2012_section
...
```

```
[lc_core_2012_section]
dynamic_path = lc_core
default_algorithms = ALL
PARAMS_GOST28147 = id-Gost28147-89-CryptoPro-A-ParamSet
```

Для использования `lc_core_2012` его идентификатор явно указывается в командной строке:

```
>lcssl req -x509 -newkey gost2001 -pkeyopt paramset:XA
-keyout ./demoCA/private/cakey.pem -out ./demoCA/cacert.pem
-engine lc_core_2012
```

6.2.12 Создание запроса на сертификат

С помощью команды req можно сгенерировать новую ключевую пару и запрос на сертификат в формате PKCS#10:

```
>lcssl req -newkey gost2001 -pkeyopt paramset:XA -out req.pem
-engine lc_core_2012
```

6.2.13 Выдача сертификата по запросу

С помощью команды ca можно на тестовом УЦ по запросу в формате PKCS#10 создать сертификат, подписанный закрытым ключом УЦ. Если для УЦ используется `lc_core_2012`, то его идентификатор явно указывается в командной строке:

```
>lcssl ca -in req.pem -out newcert.pem -engine lc_core_2012
```

6.3 Утилита store

Штатные утилиты OpenSSL рассчитаны на работу с файлами и не всегда могут быть приспособлены для работы с объектами токена. Дополнительная утилита store разработана специально для работы с объектами токена.

Порядок вызова:

```
Usage: store [options]
where options may be
-engine e          use engine e, possibly a hardware device
                   (lccryptoki_fs by default)
-genkey label      generate keypair with label friendly name
-pkeyopt keyalg:{gost2001|gost3410-2012-256|gost3410-2012-512}
         paramset:{A|B|C|XA|XB}
         key algorithm and parameter set (gost2001 and A by default)
-list              list object types and labels
-delete label      remove all objects with label friendly name
-clear             remove all objects
```

```
-rename label newlabel
                    rename all objects with label friendly name
-import label       import object with label friendly name
-inkey file         private key file path for import
-incert file        X509 certificate file path for import
-inform arg         file format for import (PEM or DER)
-inpass arg         file password for import
NB: options order may be important!
```

Например, если вызвать утилиту store с флагом -list, то будет выдан список всех объектов токена с указанием их типов и меток:

```
>store -list
Using configuration from
    C:\Program Files\LISSI-Soft\LCSSL\ssl\openssl.cnf
engine "ls11sw2012" set.
Stored objects:
1: CKO_PRIVATE_KEY, label:'pkey 1149'
2: CKO_PRIVATE_KEY, label:'my pkey'
...

OK
```

6.3.1 Генерация ключевой пары

```
>store -genkey label -pkeyopt keyalg:gost3410-2012-512 paramset:A
```

При использовании утилиты store на токене генерируется новая ключевая пара, причем ее объектам явно присваивается заданная метка label, по которой их можно будет идентифицировать в дальнейшем.

6.3.2 Импорт закрытого ключа

```
>store -import label -inkey priv_key_file
```

На токене создается объект закрытого ключа с меткой label и со значением атрибута CKA_ID, равного дайджесту SHA-1 от значения соответствующего открытого ключа. По умолчанию, предполагается, что файл закрытого ключа представлен в формате PEM. Если PEM-файл зашифрован, то нужно добавить -inpass password. Если файл представлен в формате DER, то нужно добавить -inform DER.

6.3.3 Импорт сертификата

```
>store -import label -incert cert_file
```

На токене создается объект сертификата с меткой label и со значением атрибута `CKA_ID`, равного дайджесту SHA-1 от значения открытого ключа. По умолчанию, предполагается, что файл сертификата представлен в формате PEM. Если PEM-файл зашифрован, то нужно добавить -inpass password. Если файл представлен в формате DER, то нужно добавить -inform DER.

7 Лицензии

Лицензия для OpenSSL:

```
The OpenSSL toolkit stays under a dual license, i.e. both the conditions of
the OpenSSL License and the original SSLeay license apply to the toolkit.
See below for the actual license texts. Actually both licenses are BSD-style
Open Source licenses. In case of any license issues related to OpenSSL
please contact openssl-core@openssl.org.

OpenSSL License
---------------

/* ====================================================================
 * Copyright (c) 1998-2011 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
```

```
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
 * Hudson (tjh@cryptsoft.com).
 *
 */

 Original SSLeay License
 -----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
```

```
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

В lcc2012, используется библиотека BigDigits [17] компании DI Management Servies Pty Limited в соответствии с приведенной ниже лицензией.

This source code is part of the BIGDIGITS multiple-precision arithmetic library Version 2.4 originally written by David Ireland, copyright (c) 2001-13 D.I. Management Services Pty Limited, all rights reserved. You are permitted to use compiled versions of this code at no charge as part of your own executable files and to distribute unlimited copies of such executable files for any purposes including commercial ones provided you agree to these terms and conditions and keep the copyright notices intact in the source code and you ensure that the following characters remain in any object or executable files you distribute AND clearly in any accompanying documentation:

"Contains BIGDIGITS multiple-precision arithmetic code originally written by David Ireland, copyright (c) 2001-13 by D.I. Management Services Pty Limited (www.di-mgt.com.au), and is used with permission."

David Ireland and DI Management Services Pty Limited make no representations concerning either the merchantability of this software or the suitability of this software for any particular purpose. It is provided "as is" without express or implied warranty of any kind. Our liability will be limited exclusively to the refund of the money you paid us for the software, namely nothing. By using the software you expressly agree to such a waiver. If you do not agree to the terms, do not use the software.

Please forward any comments and bug reports to ¡www.di-mgt.com.au¿. The latest version of the source code can be downloaded from ¡www.di-mgt.com.au/bigdigits.html¿.

8 Ссылки

1. Официальный сайт ООО "ЛИССИ-Софт". – http://soft.lissi.ru/.

2. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. – http://protect.gost.ru/document.aspx?control=7&id=139177.

3. ГОСТ Р 34.10-2001. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. – http://protect.gost.ru/document.aspx?control=7&id=131131.

4. ГОСТ Р 34.10-2012. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. – Москва, Стандартинформ, 2012.

5. ГОСТ Р 34.11-94. Информационная технология. Криптографическая защита информации. Функция хеширования. – http://protect.gost.ru/document.aspx?control=7&id=134550.

6. ГОСТ Р 34.11-2012. Информационная технология. Криптографическая защита информации. Функция хеширования. – Москва, Стандартинформ, 2012.

7. RFC 4357. V. Popov, I. Kurepkin, S. Leontiev. Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms. – CRYPTO-PRO, January 2006. – http://www.ietf.org/rfc/rfc4357.txt.

8. RFC 4490. S. Leontiev, Ed., G. Chudov, Ed. Using the GOST 28147-89, GOST R 34.11-94, GOST R 34.10-94, and GOST R 34.10-2001 Algorithms with Cryptographic Message Syntax (CMS). – CRYPTO-PRO, May 2006. – http://tools.ietf.org/html/rfc4490.

9. Методические рекомендации по заданию узлов замены блока подстановки алгоритма шифрования ГОСТ 28147-89 (готовится к публикации). – Москва, ТК 26, 2013.

10. Методические рекомендации по заданию параметров эллиптических кривых в соответствии с ГОСТ Р 34.10-2012 (готовится к публикации). – Москва, ТК 26, 2013.

11. Методические рекомендации по криптографическим алгоритмам, сопутствующим применению стандартов ГОСТ Р 34.10-2012 и ГОСТ Р 34.11-2012 (готовится к публикации). – Москва, ТК 26, 2013.

12. Парольная защита с использованием алгоритмов ГОСТ. Дополнения к PKCS#5 (готовится к публикации). – Москва, ТК 26, 2013.

13. Расширение PKCS#11 для использования российских криптографических алгоритмов. – Технический комитет по стандартизации (ТК 26) "Криптографическая защита информации". – Москва, ТК 26, 2008.

14. Расширение PKCS#11 для использования российских стандартов ГОСТ Р 34.10-2012 и ГОСТ Р 34.11-2012 (готовится к публикации). – Технический комитет по стандартизации (ТК 26) "Криптографическая защита информации". – Москва, ТК 26, 2013.

15. Официальный сайт проекта OpenSSL. – http://www.openssl.org/.

16. PKCS#11 v2.30: Cryptographic Token Interface Standard. - RSA Laboratories, 2009. - http://www.rsa.com/rsalabs/node.asp?id=2133.

17. BigDigits multiple-precision arithmetic source code. – http://www.di-mgt.com.au/bigdigits.html.

18. Кроссплатформенная сборочная система CMake. – http://www.cmake.org/.