

LS11SW - программная реализация
PKCS#11
с поддержкой ГОСТ Р34.10-2012
и ГОСТ Р34.11-2012
Руководство Программиста



11 апреля 2014 г.

Оглавление

1	Введение	4
1.1	Обзор архитектуры	4
1.2	Системные требования	4
1.3	Особенности реализации	4
2	Основные сведения	7
2.1	Состав и структура	7
2.1.1	Библиотека ls11sw2012	7
2.1.2	Поддерживаемые типы объектов	7
2.1.3	Поддерживаемые механизмы	8
2.2	Установка	9
2.3	Создание программного токена	10
2.4	Утилита конфигурации	10
2.4.1	Получение информации о библиотеке	11
2.4.2	Получение информации о слотах	11
2.4.3	Получение информации о токене	11
2.4.4	Инициализация токена	12
2.4.5	Назначение PIN администратора безопасности	12
2.4.6	Инициализация пользовательского PIN	13
2.4.7	Изменение пользовательского PIN	13
3	Программирование	14
3.1	Функции общего назначения	14
3.2	Инициализация библиотеки	37
3.3	Примеры программ	37
3.3.1	Получение информации о слотах и токенах	37
3.3.2	Работа с объектами данных	39
3.3.3	Работа с сертификатами	44
3.3.4	Генерация дайджеста	49
3.3.5	Генерация HMAC	89
3.3.6	Генерация ключевой пары, ЭЦП и ее проверка	129
3.3.7	Генерация ключей согласования	140
3.3.8	Шифрование по ГОСТ 28147-89	185
3.3.9	Шифрование в режиме ECB	200
3.3.10	Шифрование в режиме CBC	210
3.3.11	Шифрование в режиме CNT	217

3.3.12	Генерация имитовставки	231
3.3.13	PKCS#8 и вывод открытого ключа по закрытому	240
3.3.14	Шифрование ключей по ГОСТ 28147-89	258
3.3.15	Шифрование ключей по ГОСТ Р34.10-2001	276
3.3.16	Генерация ключа шифрования на пароле	308
3.3.17	Генерация ключа аутентификации на пароле	318
3.3.18	Механизмы для TLS	322
3.3.19	Использование нескольких потоков	350
4	Лицензии	358
4.1	Лицензия для BigDigits	358
5	Ссылки	359

1 Введение

LS11SW является программной реализацией ООО "ЛИССИ-Софт" [1] стандарта PKCS#11 API [16], дополненного поддержкой российских криптографических алгоритмов в соответствии со спецификациями, выработанными Техническим комитетом по стандартизации (ТК 26) "Криптографическая защита информации" [2, 14]. Начиная с версии 5.0, LS11SW поддерживает алгоритмы ГОСТ Р34.10-2012, ГОСТ Р34.11-2012, а также сопутствующие алгоритмы и параметры, определенные руководящими документами ТК 26.

Поддержка алгоритмов российской криптографии в LS11SW осуществляется библиотекой с интерфейсом PKCS#11. LS11SW позволяет работать только с одним программным токеном, который располагается в предопределенном месте файловой системы пользователя.

1.1 Обзор архитектуры

LS11SW предоставляет библиотеку PKCS#11, динамически подключаемую приложением. При инициализации библиотеки создается единственный слот для подключения уже созданного программного токена. Программный токен создается в файловой системе пользователя при его лицензировании.

LS11SW API включает функции, описанные в спецификации PKCS#11 API [16], с учетом расширения алгоритмами российской криптографии [14].

1.2 Системные требования

LS11SW реализована кросс-платформенным образом, так что она, в принципе, может быть портирована в любую операционную систему, где поддерживается язык Си. Текущая версия работает в операционных системах Windows, Linux и Mac OS X на 32-х и 64-х разрядных платформах. Имеется также вариант библиотеки для мобильных устройств с операционной системой Android.

1.3 Особенности реализации

В LS11SW поддерживаются все российские криптографические механизмы, определенные в [14]. Все криптографические механизмы поддерживаются библиотекой программно. Кроме того, библиотека позволяет работать с временными сессионными объектами для всех поддерживаемых механизмов.

LS11SW поддерживает создание и использование объектов параметров домена при симметричном шифровании. Если при шифровании в ключе задан OID доступного для текущей сессии объекта параметров домена, то будут использованы параметры из этого объекта. Если же такого объекта нет, то будет предпринята попытка найти и использовать параметры для данного OID среди предопределенных наборов параметров, заданных в RFC 4357[8]. Таким образом, наличие в сессии соответствующих объектов параметров домена не является обязательным, если используются предопределенные наборы параметров. Заметим, что согласно RFC 4357[8], в объектах параметров домена могут быть заданы только режимы шифрования CNT(0), CFB(1) или CBC(2).

В LS11SW в качестве альтернативного варианта реализован также нестандартный механизм СКМ_GOST28147_CNT, позволяющий использовать режим шифрования CNT с предопределенными наборами параметров, не применяя объекты параметров домена.

Заметим, что при шифровании в блочных режимах ECB и CBC исходные данные должны поступать порциями, длина которых кратна размеру блока, т.е. 8-ми байтам. Организация паддинга при этом возлагается на прикладную программу.

Для механизмов ГОСТ Р34.10-2001, ГОСТ Р34.10-2012, ГОСТ Р 34.11-94 разрешается использовать только предопределенные наборы параметров, определенные в RFC 4357 и в документах ТК 26. Объекты параметров домена при этом не используются. Алгоритмы ГОСТ Р 34.11-2012 вообще не имеют параметров, поэтому вместо OID параметров в соответствующих атрибутах указывается OID алгоритма.

Если при генерации ключевой пары в шаблоне закрытого ключа не был задан атрибут СКА_ID, то он автоматически устанавливается равным SHA-1 от значения открытого ключа.

В интересах поддержки российской реализации стандарта PKCS#12 дополнительно реализован механизм СКМ_PBA_GOSTR3411_WITH_GOSTR3411_HMAC, используемый для генерации ключа на пароле при выработке и проверке дайджеста хранилища. Кроме того, добавлен механизм СКМ_GOST28147_PKCS8_KEY_WRAP для упаковки и распаковки закрытого ключа ГОСТ Р 34.10-2001 по стандарту PKCS#8 с помощью шифрования секретным ключом, генерируемом на пароле механизмом СКМ_PKCS5_PBKDF2.

Функции генерации случайных чисел поддерживаются встроенным в библиотеку криптографическим генератором.

Библиотека поддерживает работу из нескольких потоков, однако по стандарту нельзя использовать одну и ту же сессию из разных потоков.

Количество одновременно открытых сессий в приложении не должно превышать 256.

Создание и лицензирование токена производится через web-интерфейс на сайте "ЛИССИ-Софт"[1]. После лицензирования на токене нужно установить метку и PIN пользователя. Его дальнейшую инициализацию с присвоением метки и установку PIN можно выполнить либо утилитой p11conf, либо программно.

Программный токен считается не извлекаемым, поэтому никакие события не слоте библиотекой не регистрируются. В связи с этим, функция C_WaitForSlotEvent

никогда не возвращает `СКР_ОК`. Если при вызове задан флаг `СКФ_DONT_BLOCK`, то сразу возвращается `СКР_NO_EVENT`. Если функция вызвана до `C_Initialize` или `C_Finalize` вызвана при заблокированной `C_WaitForSlotEvent`, то `C_WaitForSlotEvent` сразу завершает работу и возвращает `СКР_КРИПТОКИ_NOT_INITIALIZED`. Наконец, если программный токен не лицензирован, то возвращается `СКР_TOKEN_NOT_RECOGNIZED`.

2 Основные сведения

2.1 Состав и структура

В состав LS11SW входят:

- Библиотека PKCS#11 ls11sw2012
- Утилита конфигурации p11conf
- Набор тестовых примеров
- Руководство программиста

Имена файлов динамических библиотек и выполняемых файлов в разных операционных системах отличаются. Например, в Windows имя файла библиотеки ls11sw2012 - ls11sw2012.dll, в Linux - libls11sw2012.so, в Mac OS X - libls11sw2012.dylib. Это отличие нужно учитывать при загрузке библиотеки из прикладной программы. Выполняемый файл утилиты p11conf в Windows называется p11conf.exe, а в Linux и Mac OS X - просто p11conf.

Набор тестовых примеров поставляется в виде CMake-проекта [18]. Для генерации сборочного проекта на целевой платформе нужно скачать и установить последнюю версию CMake, затем из папки build выполнить команду

```
сmake ..
```

В результате, в папке build будут созданы файлы сборочного проекта, после чего нужно продолжить сборку тестов уже для этого проекта. Запуск всех тестов на выполнение можно произвести командой ctest.

2.1.1 Библиотека ls11sw2012

Библиотека PKCS#11 API ls11sw2012 должна быть загружена прикладной программой динамически, после чего из нее извлекается список функций PKCS#11.

Затем вызывается функция C_Initialize, с которой начинается работа с библиотекой по стандарту PKCS#11.

2.1.2 Поддерживаемые типы объектов

Программный токен LS11SW в слоте 0 может хранить объекты ключевой пары (СКО_PRIVATE_KEY, СКО_PUBLIC_KEY) типа СКК_GOSTR3410 или СКК_GOSTR3410_512,

объекты сертификатов CKO_CERTIFICATE типа X.509, объекты данных CKO_DATA, ключи для симметричного шифрования и хеширования, а также параметры домена.

На том же нулевом слоте могут быть созданы временные сессионные объекты перечисленных типов.

2.1.3 Поддерживаемые механизмы

LS11SW поддерживает следующие механизмы PKCS#11:

- CKM_GOST28147_KEY_GEN
- CKM_GOST28147
- CKM_GOST28147_KEY_WRAP
- CKM_GOST28147_ECB
- CKM_GOST28147_CNT
- CKM_GOST28147_MAC
- CKM_GOST28147_KEY_CPDIVERSIFY
- CKM_GOST28147_PKCS8_KEY_WRAP
- CKM_GOSTR3411
- CKM_GOSTR3411_12_256
- CKM_GOSTR3411_12_512
- CKM_GOSTR3411_HMAC
- CKM_GOSTR3411_12_256_HMAC
- CKM_GOSTR3411_12_512_HMAC
- CKM_GOSTR3411_12_256_HMAC_KDF
- CKM_GOSTR3410_KEY_PAIR_GEN
- CKM_GOSTR3410_512_KEY_PAIR_GEN
- CKM_GOSTR3410
- CKM_GOSTR3410_512
- CKM_GOSTR3410_WITH_GOSTR3411
- CKM_GOSTR3410_WITH_GOSTR3411_12_256
- CKM_GOSTR3410_WITH_GOSTR3411_12_512

- CKM_GOSTR3410_DERIVE
- CKM_GOSTR3410_12_DERIVE
- CKM_GOSTR3410_KEY_WRAP
- CKM_GOSTR3410_PUBLIC_KEY_DERIVE
- CKM_PKCS5_PBKD2
- CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC
- CKM_TLS_GOST_KEY_AND_MAC_DERIVE
- CKM_TLS_GOST_PRE_MASTER_KEY_GEN
- CKM_TLS_GOST_MASTER_KEY_DERIVE
- CKM_TLS_GOST_PRF
- CKM_SHA_1
- CKM_MD5

Согласно стандарту PKCS#11, значения полей `ulMinKeySize` и `ulMaxKeySize` в структуре `CK_MECHANISM_INFO` не используются для механизмов ГОСТ. В данной реализации значения этих полей содержат для механизмов ГОСТ Р34.10 размер закрытого ключа в битах (256 или 512), для механизмов ГОСТ 28147-89 - размер ключа шифрования в байтах (32), для механизма `CKM_TLS_GOST_MASTER_KEY_DERIVE` - размер мастер-ключа в байтах (48). Все эти значения имеют чисто информационный характер и не используются в алгоритмах библиотеки.

2.2 Установка

Библиотека `ls11sw2012` устанавливается в целевую папку. Кроме того, туда же устанавливается утилита конфигурации `r11conf`.

Важное замечание. На целевой платформе нужно обеспечить прикладным программам возможность найти библиотеки и утилиты по именам путем добавления полного пути к содержащей их папке к значению переменной среды `PATH` или `LD_LIBRARY_PATH`, в зависимости от операционной системы.

Документация и тестовые примеры для `LS11SW` скачиваются отдельно с сайта "ЛИССИ-Софт" [1].

2.3 Создание программного токена

Программный токен создается в predetermined месте файловой системы пользователя в процессе диалога с сервером через web-интерфейс на сайте "ЛИССИ-Софт"[1]. Этому программному токenu присваивается уникальный серийный номер, под которым он регистрируется на сервере в соответствии с его лицензией. С данным программным токеном сможет работать только тот пользователь, который его создавал, причем только на том же компьютере и в той же операционной системе.

Для размещения файлов программного токена в системе Windows используется папка %USERPROFILE%\LS11SW в файловом пространстве пользователя. В системе Linux для той же цели в домашнем каталоге пользователя используется папка .LS11SW.

Программный токен первоначально создается не инициализированным, с пустой меткой и без PIN пользователя. Инициализация токена с присвоением метки и назначение PIN пользователя могут быть выполнены с помощью web-интерфейса, утилиты конфигурации p11conf (см. раздел "Утилита конфигурации") или программно функциями PKCS#11. Для созданного токена устанавливается умалчиваемое значение PIN администратора безопасности (Security Officer или сокращенно SO) – 87654321.

Заметим, что для доступа к программному токenu в файловом пространстве пользователя SO должен работать в сеансе пользователя токена. Фактически это означает, что для программного токена LS11SW SO и пользователь токена – это одно и то же лицо, хотя значения PIN у SO и пользователя могут быть разными.

Если по какой-то причине файлы программного токена оказались поврежденными, то папку программного токена можно удалить и провести лицензирование заново. Поскольку в базе данных сервера уже имеется запись о предыдущем лицензировании, то повторное лицензирование не потребует оплаты.

2.4 Утилита конфигурации

LS11SW предоставляет утилиту командной строки p11conf для конфигурирования и администрирования токенов, поддерживаемых в системе. Утилита работает исключительно через API, поэтому ее можно применять для работы с любыми библиотеками PKCS#11, а не только с ls11sw.

Заметим, что с флагом -s задается идентификатор слота. Этот идентификатор необходимо задавать, когда операция выполняется с конкретным токеном. Идентификаторы всех слотов можно получить с флагом -s.

Утилита предоставляет возможности, о которых сообщается при запуске команды p11conf -h. К ним относятся инициализация токена, инициализация и изменение PIN администратора безопасности, инициализация и изменение PIN пользователя и др. Следующие примеры показывают опции утилиты p11conf и выдачу информации о слотах в системе до инициализации токенов.

```
> ./p11conf -h
```

```
usage: p11conf [-hitsmIupPred] -A APIpath [-c slotID
-U userPin -S S0Pin -n newPin -L label]
    -h display usage
    -i display PKCS11 info
    -t display token info
    -s display slot info
    -m display mechanism list
    -I initialize token
    -u initialize user PIN
    -p set the user PIN
    -P set the S0 PIN
    -r remove all objects
    -e enumerate objects
    -d dump enumerated object attributes
```

2.4.1 Получение информации о библиотеке

```
> ./p11conf -A ls11sw2012 -i
PKCS#11 Info
    Version 2.30
    Manufacturer: LISSI-Soft
    Flags: 0x0
    Library Description: ls11sw2012 PKCS#11 library
    Library Version 5.0
OK
```

Данная программа также позволяет выполнять некоторые простые запросы слотам и токенам, например для получения информации о слотах, токенах и для получения списка поддерживаемых механизмов.

2.4.2 Получение информации о слотах

```
> ./p11conf -A ls11sw2012 -s
Slot ID 0 Info
    Description: LS11SW Slot 0
    Manufacturer: LISSI-Soft
    Flags: 0x7 (TOKEN_PRESENT)
    Hardware Version: 1.0
    Firmware Version: 1.0
OK
```

2.4.3 Получение информации о токене

```
> ./p11conf -A ls11sw2012 -t -c 0
Token #0 Info:
```

```
Label: Token Label
Manufacturer: LISSI-Soft
Model: LS11SW
Serial Number: 1234567887654321
Flags: 0x40C (LOGIN_REQUIRED|USER_PIN_INITIALIZED|TOKEN_INITIALIZED)
Sessions: 0/256
R/W Sessions: 0/256
PIN Length: 4-32
Public Memory: 0xFFFFFFFF/0xFFFFFFFF
Private Memory: 0xFFFFFFFF/0xFFFFFFFF
Hardware Version: 1.0
Firmware Version: 1.0
Time: 18:58:42
```

OK

Программный токен создается не инициализированным. Прежде чем приложение сможет использовать новый токен LS11SW, нужно выполнить следующие начальные шаги (каждому шагу соответствует пример кода):

2.4.4 Инициализация токена

Перед использованием токена он должен быть однажды инициализирован. Ключевой частью данного процесса является назначение токену уникальной метки (имени). Для этого понадобится PIN администратора безопасности (SO) для данного токена (начальное значение SO PIN для только что созданного токена LS11SW – 87654321).

Замечание. Все значения PIN отображаются звездочками при вводе.

```
> ./p11conf -A ls11sw2012 -I -c 0
Enter the SO PIN: 87654321
Enter a unique token label: LissiSW
```

То же самое можно выполнить, задавая значения SO PIN и метки прямо в командной строке:

```
> ./p11conf -A ls11sw2012 -I -c 0 -S 87654321 -L LissiSW
```

2.4.5 Назначение PIN администратора безопасности

Правильной организационной практикой является изменение администратором безопасности своего PIN сразу после инициализации токена. Данная процедура предотвращает возможность инициализировать токен посторонним лицам и удалить тем самым все созданные объекты (например, ключи и сертификаты).

```
> ./p11conf -A ls11sw2012 -P -c 0
```

```
Enter the SO PIN: 87654321
Enter the new SO PIN: 76543210
Re-enter the new SO PIN: 76543210
```

Вариант ввода в командной строке:

```
> ./p11conf -A ls11sw2012 -P -c 0 -S 87654321 -n 76543210
```

PIN не обязательно должен быть цифровым, допускаются любые символы, но во избежание проблем с кодировками рекомендуется использовать латинскую половину кодовой таблицы.

2.4.6 Инициализация пользовательского PIN

Данная операция выполняется администратором безопасности перед передачей токена пользователю. Программный токен изначально создается в файловом пространстве пользователя, однако формальные требования стандарта должны быть выполнены и для него.

```
> ./p11conf -A ls11sw2012 -u -c 0
Enter the SO PIN: 76543210
Enter the new user PIN: 12345678
Re-enter the new user PIN: 12345678
```

Вариант ввода в командной строке:

```
> ./p11conf -A ls11sw2012 -u -c 0 -S 76543210 -n 12345678
```

2.4.7 Изменение пользовательского PIN

Первое, что должен сделать пользователь после получения токена от администратора безопасности, - это изменение PIN.

```
> ./p11conf -A ls11sw2012 -p -c 0
Enter user PIN: 12345678
Enter the new user PIN: 01234567
Re-enter the new user PIN: 01234567
```

Вариант ввода в командной строке:

```
> ./p11conf -A ls11sw2012 -p -c 0 -U 12345678 -n 01234567
```

Если значение SO PIN для токена потеряно, а токен заблокирован из-за нескольких вводов неверного PIN, то штатными средствами LS11SW разблокировать токен нельзя из соображений секретности его данных — его можно только создать заново.

Замечание. Начальное значение USER PIN, установленное SO, запрещается в дальнейшем устанавливать пользователю из соображений безопасности.

3 Программирование

Все приведенные ниже примеры содержатся в папке проекта tests. Для удобства сборки примеров там же содержится сборочный файл CMakeLists.txt для генерации тестового проекта с помощью CMake [18]. Войдите в папку tests/build и выполните из командной строки команду:

```
>cmake ..
```

В результате выполнения команды в папке tests/build генерируются проектные файлы для вашей сборочной системы. Например, для MSVS будут созданы проектные файлы для решения ls11sw_tests.sln, а для Linux - соответствующий make-файл. Дальнейшую сборку производите уже вашей сборочной системой.

Все сборочные промежуточные и результирующие файлы будут созданы в папке tests/build.

3.1 Функции общего назначения

Во всех приведенных далее примерах используются функции общего назначения, представленные в файлах test_common.h и test_common.c. Здесь приводятся исходные тексты этих файлов, чтобы было понятно, для чего эти функции используются в тестовых примерах.

Листинг 3.1: test_common.h

```
#pragma once
#ifdef WIN32
#include <windows.h>
#include <conio.h>
#else
#include <unistd.h>
#include <dlfcn.h>
#include <termios.h>
#include <strings.h>
#include <nl_types.h>
#include <sys/time.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <memory.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include "pkcs11ctrl.h"

#ifdef WIN32
#define PKCS11_API_PATH "ls11sw2012"
#elif __APPLE__
#define PKCS11_API_PATH "libls11sw2012.dylib"
#else
#define PKCS11_API_PATH "libls11sw2012.so"
#endif

extern CK_CHAR *api_path;
extern CK_CHAR *user_pin;
extern CK_UTF8CHAR *so_pin;

#define PIN_SIZE 80
#define BACK_SPACE '\b'
#define LINE_FEED '\n'
#define CARRIAGE_RETURN '\r'
#define END_OF_TEXT 0x03 // ETX (Ctrl-C)

#ifdef WIN32
extern HMODULE dll;
#else
extern void *dll;
#endif
extern CK_FUNCTION_LIST_PTR funcs;
extern CK_C_Ctrl pctrl;
extern CK_C_INITIALIZE_ARGS *pinit_args_os_locking;
extern CK_C_INITIALIZE_ARGS *pinit_args_app_locking;
extern CK_SLOT_ID SlotId;
extern CK_SLOT_ID_PTR SlotList;
extern CK_ULONG SlotCount;

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

long process_time(struct timeb t1, struct timeb t2);
void show_error(char *str, CK_RV rc);
void process_ret_code(CK_RV rc);
void print_hex(CK_BYTE *buf, CK_ULONG len);
int print_bytes(CK_BYTE *buf, CK_ULONG len, char *str);
```

```

CK_RV AppCreateMutex( void **pmutex );
CK_RV AppDestroyMutex( void *mutex );
CK_RV AppLockMutex( void *mutex );
CK_RV AppUnlockMutex( void *mutex );
CK_RV get_function_list( CK_CHAR *api_path , CK_FUNCTION_LIST_PTR
    *pfuncs );
CK_RV init( CK_CHAR *api_path ,
    CK_FUNCTION_LIST_PTR *pfuncs ,
    CK_C_INITIALIZE_ARGS *pinit_args );
CK_RV cleanup( CK_FUNCTION_LIST_PTR func );
CK_RV get_slot_list( CK_FUNCTION_LIST_PTR func ,
    CK_SLOT_ID **ppSlotList ,
    CK_ULONG *pulSlotCount );
CK_RV find_slot_with_token( CK_FUNCTION_LIST_PTR func ,
    CK_SLOT_ID *pSlotList ,
    CK_ULONG ulSlotCount ,
    CK_SLOT_ID *pSlotId );
char *get_mechanism_name( CK_MECHANISM_TYPE );
CK_RV display_pkcs11_info( CK_FUNCTION_LIST_PTR func );
CK_RV display_slot_info( CK_FUNCTION_LIST_PTR func , CK_SLOT_ID
    SlotId );
CK_RV display_token_info( CK_FUNCTION_LIST_PTR func , CK_SLOT_ID
    SlotId );
CK_RV display_mechanisms( CK_FUNCTION_LIST_PTR func , CK_SLOT_ID
    slot_id );
CK_RV pin_callback(
    CK_SESSION_HANDLE hSession ,
    CK_NOTIFICATION event , // CKN_LISSI_PIN_CALLBACK
    CK_VOID_PTR pApplication // output PIN buffer here
);
CK_RV test_main( int argc , char *argv [] );
CK_RV check_value( CK_BYTE *value , CK_ULONG len , CK_BYTE *eth ,
    CK_ULONG eth_len );

#define CHECK(v, len, ethalon) \
do { \
    if ((rc = check_value(v, len, ethalon, sizeof(ethalon))) != \
        CKR_OK) { return rc; } \
} while (0)

```

Листинг 3.2: test_common.c

```
#include "test_common.h"
```



```
#pragma warning(disable : 4996)

CK_FUNCTION_LIST_PTR funcs = NULL;
CK_C_Ctrl pctrl = NULL;
#ifdef WIN32
HMODULE dll;
#else
void *dll;
#endif
CK_UTF8CHAR *user_pin = "01234567";
CK_UTF8CHAR *so_pin = "76543210";
CK_CHAR *api_path = PKCS11_API_PATH;
CK_SLOT_ID SlotId;
CK_SLOT_ID_PTR SlotList = NULL;
CK_ULONG SlotCount = 0;

CK_C_INITIALIZE_ARGS cinit_args_os_locking = {
    NULL,
    NULL,
    NULL,
    NULL,
    CKF_OS_LOCKING_OK,
    NULL};
CK_C_INITIALIZE_ARGS cinit_args_app_locking = {
    AppCreateMutex,
    AppDestroyMutex,
    AppLockMutex,
    AppUnlockMutex,
    0,
    NULL};
CK_C_INITIALIZE_ARGS *pinit_args_os_locking =
    &cinit_args_os_locking;
CK_C_INITIALIZE_ARGS *pinit_args_app_locking =
    &cinit_args_app_locking;

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

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

```
ms += (long)(s*1000);
return ms;
}

void process_ret_code( CK_RV rc )
{
    switch (rc) {
        case CKR_OK:
            printf(" CKR_OK"); break;
        case CKR_CANCEL:
            printf(" CKR_CANCEL"); break;
        case CKR_HOST_MEMORY:
            printf(" CKR_HOST_MEMORY"); break;
        case CKR_SLOT_ID_INVALID:
            printf(" CKR_SLOT_ID_INVALID"); break;
        case CKR_GENERAL_ERROR:
            printf(" CKR_GENERAL_ERROR"); break;
        case CKR_FUNCTION_FAILED:
            printf(" CKR_FUNCTION_FAILED"); break;
        case CKR_ARGUMENTS_BAD:
            printf(" CKR_ARGUMENTS_BAD"); break;
        case CKR_NO_EVENT:
            printf(" CKR_NO_EVENT"); break;
        case CKR_NEED_TO_CREATE_THREADS:
            printf(" CKR_NEED_TO_CREATE_THREADS"); break;
        case CKR_CANT_LOCK:
            printf(" CKR_CANT_LOCK"); break;
        case CKR_ATTRIBUTE_READ_ONLY:
            printf(" CKR_ATTRIBUTE_READ_ONLY"); break;
        case CKR_ATTRIBUTE_SENSITIVE:
            printf(" CKR_ATTRIBUTE_SENSITIVE"); break;
        case CKR_ATTRIBUTE_TYPE_INVALID:
            printf(" CKR_ATTRIBUTE_TYPE_INVALID"); break;
        case CKR_ATTRIBUTE_VALUE_INVALID:
            printf(" CKR_ATTRIBUTE_VALUE_INVALID"); break;
        case CKR_DATA_INVALID:
            printf(" CKR_DATA_INVALID"); break;
        case CKR_DATA_LEN_RANGE:
            printf(" CKR_DATA_LEN_RANGE"); break;
        case CKR_DEVICE_ERROR:
            printf(" CKR_DEVICE_ERROR"); break;
        case CKR_DEVICE_MEMORY:
            printf(" CKR_DEVICE_MEMORY"); break;
        case CKR_DEVICE_REMOVED:

```

```
    printf(" CKR_DEVICE_REMOVED"); break;
case CKR_ENCRYPTED_DATA_INVALID:
    printf(" CKR_ENCRYPTED_DATA_INVALID"); break;
case CKR_ENCRYPTED_DATA_LEN_RANGE:
    printf(" CKR_ENCRYPTED_DATA_LEN_RANGE"); break;
case CKR_FUNCTION_CANCELED:
    printf(" CKR_FUNCTION_CANCELED"); break;
case CKR_FUNCTION_NOT_PARALLEL:
    printf(" CKR_FUNCTION_NOT_PARALLEL"); break;
case CKR_FUNCTION_NOT_SUPPORTED:
    printf(" CKR_FUNCTION_NOT_SUPPORTED"); break;
case CKR_KEY_HANDLE_INVALID:
    printf(" CKR_KEY_HANDLE_INVALID"); break;
case CKR_KEY_SIZE_RANGE:
    printf(" CKR_KEY_SIZE_RANGE"); break;
case CKR_KEY_TYPE_INCONSISTENT:
    printf(" CKR_KEY_TYPE_INCONSISTENT"); break;
case CKR_KEY_NOT_NEEDED:
    printf(" CKR_KEY_NOT_NEEDED"); break;
case CKR_KEY_CHANGED:
    printf(" CKR_KEY_CHANGED"); break;
case CKR_KEY_NEEDED:
    printf(" CKR_KEY_NEEDED"); break;
case CKR_KEY_INDIGESTIBLE:
    printf(" CKR_KEY_INDIGESTIBLE"); break;
case CKR_KEY_FUNCTION_NOT_PERMITTED:
    printf(" CKR_KEY_FUNCTION_NOT_PERMITTED"); break;
case CKR_KEY_NOT_WRAPPABLE:
    printf(" CKR_KEY_NOT_WRAPPABLE"); break;
case CKR_KEY_UNEXTRACTABLE:
    printf(" CKR_KEY_UNEXTRACTABLE"); break;
case CKR_MECHANISM_INVALID:
    printf(" CKR_MECHANISM_INVALID"); break;
case CKR_MECHANISM_PARAM_INVALID:
    printf(" CKR_MECHANISM_PARAM_INVALID"); break;
case CKR_OBJECT_HANDLE_INVALID:
    printf(" CKR_OBJECT_HANDLE_INVALID"); break;
case CKR_OPERATION_ACTIVE:
    printf(" CKR_OPERATION_ACTIVE"); break;
case CKR_OPERATION_NOT_INITIALIZED:
    printf(" CKR_OPERATION_NOT_INITIALIZED"); break;
case CKR_PIN_INCORRECT:
    printf(" CKR_PIN_INCORRECT"); break;
case CKR_PIN_INVALID:
```

```
    printf(" CKR_PIN_INVALID"); break;
case CKR_PIN_LEN_RANGE:
    printf(" CKR_PIN_LEN_RANGE"); break;
case CKR_PIN_EXPIRED:
    printf(" CKR_PIN_EXPIRED"); break;
case CKR_PIN_LOCKED:
    printf(" CKR_PIN_LOCKED"); break;
case CKR_SESSION_CLOSED:
    printf(" CKR_SESSION_CLOSED"); break;
case CKR_SESSION_COUNT:
    printf(" CKR_SESSION_COUNT"); break;
case CKR_SESSION_HANDLE_INVALID:
    printf(" CKR_SESSION_HANDLE_INVALID"); break;
case CKR_SESSION_PARALLEL_NOT_SUPPORTED:
    printf(" CKR_SESSION_PARALLEL_NOT_SUPPORTED"); break;
case CKR_SESSION_READ_ONLY:
    printf(" CKR_SESSION_READ_ONLY"); break;
case CKR_SESSION_EXISTS:
    printf(" CKR_SESSION_EXISTS"); break;
case CKR_SESSION_READ_ONLY_EXISTS:
    printf(" CKR_SESSION_READ_ONLY_EXISTS"); break;
case CKR_SESSION_READ_WRITE_SO_EXISTS:
    printf(" CKR_SESSION_READ_WRITE_SO_EXISTS"); break;
case CKR_SIGNATURE_INVALID:
    printf(" CKR_SIGNATURE_INVALID"); break;
case CKR_SIGNATURE_LEN_RANGE:
    printf(" CKR_SIGNATURE_LEN_RANGE"); break;
case CKR_TEMPLATE_INCOMPLETE:
    printf(" CKR_TEMPLATE_INCOMPLETE"); break;
case CKR_TEMPLATE_INCONSISTENT:
    printf(" CKR_TEMPLATE_INCONSISTENT"); break;
case CKR_TOKEN_NOT_PRESENT:
    printf(" CKR_TOKEN_NOT_PRESENT"); break;
case CKR_TOKEN_NOT_RECOGNIZED:
    printf(" CKR_TOKEN_NOT_RECOGNIZED"); break;
case CKR_TOKEN_WRITE_PROTECTED:
    printf(" CKR_TOKEN_WRITE_PROTECTED"); break;
case CKR_UNWRAPPING_KEY_HANDLE_INVALID:
    printf(" CKR_UNWRAPPING_KEY_HANDLE_INVALID"); break;
case CKR_UNWRAPPING_KEY_SIZE_RANGE:
    printf(" CKR_UNWRAPPING_KEY_SIZE_RANGE"); break;
case CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT:
    printf(" CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT"); break;
case CKR_USER_ALREADY_LOGGED_IN:
```

```
    printf(" CKR_USER_ALREADY_LOGGED_IN"); break;
case CKR_USER_NOT_LOGGED_IN:
    printf(" CKR_USER_NOT_LOGGED_IN"); break;
case CKR_USER_PIN_NOT_INITIALIZED:
    printf(" CKR_USER_PIN_NOT_INITIALIZED"); break;
case CKR_USER_TYPE_INVALID:
    printf(" CKR_USER_TYPE_INVALID"); break;
case CKR_USER_ANOTHER_ALREADY_LOGGED_IN:
    printf(" CKR_USER_ANOTHER_ALREADY_LOGGED_IN"); break;
case CKR_USER_TOO_MANY_TYPES:
    printf(" CKR_USER_TOO_MANY_TYPES"); break;
case CKR_WRAPPED_KEY_INVALID:
    printf(" CKR_WRAPPED_KEY_INVALID"); break;
case CKR_WRAPPED_KEY_LEN_RANGE:
    printf(" CKR_WRAPPED_KEY_LEN_RANGE"); break;
case CKR_WRAPPING_KEY_HANDLE_INVALID:
    printf(" CKR_WRAPPING_KEY_HANDLE_INVALID"); break;
case CKR_WRAPPING_KEY_SIZE_RANGE:
    printf(" CKR_WRAPPING_KEY_SIZE_RANGE"); break;
case CKR_WRAPPING_KEY_TYPE_INCONSISTENT:
    printf(" CKR_WRAPPING_KEY_TYPE_INCONSISTENT"); break;
case CKR_RANDOM_SEED_NOT_SUPPORTED:
    printf(" CKR_RANDOM_SEED_NOT_SUPPORTED"); break;
case CKR_RANDOM_NO_RNG:
    printf(" CKR_RANDOM_NO_RNG"); break;
case CKR_BUFFER_TOO_SMALL:
    printf(" CKR_BUFFER_TOO_SMALL"); break;
case CKR_SAVED_STATE_INVALID:
    printf(" CKR_SAVED_STATE_INVALID"); break;
case CKR_INFORMATION_SENSITIVE:
    printf(" CKR_INFORMATION_SENSITIVE"); break;
case CKR_STATE_UNSAVEABLE:
    printf(" CKR_STATE_UNSAVEABLE"); break;
case CKR_CRYPTOKI_NOT_INITIALIZED:
    printf(" CKR_CRYPTOKI_NOT_INITIALIZED"); break;
case CKR_CRYPTOKI_ALREADY_INITIALIZED:
    printf(" CKR_CRYPTOKI_ALREADY_INITIALIZED"); break;
case CKR_MUTEX_BAD:
    printf(" CKR_MUTEX_BAD"); break;
case CKR_MUTEX_NOT_LOCKED:
    printf(" CKR_MUTEX_NOT_LOCKED"); break;
}
}
```

```
void show_error( char *str, CK_RV rc )
{
    printf("%s returned: 0x%x", str, rc );
    process_ret_code( rc );
    printf("\n");
}

void print_hex( CK_BYTE *buf, CK_ULONG len )
{
    CK_ULONG i = 0, j;

    while ( i < len ) {
        for ( j=0; (j < 8) && (i < len); j++, i++)
            printf("0x%02x, ", buf[i] & 0xff);
        printf("\n");
    }
    printf("\n");
}

int print_bytes(unsigned char *buf, unsigned long len, char *str
)
{
    unsigned long i, j;
    for (i=0, j=0; i<len; i++) {
        if (i!=0 && i%8==0) {
            j += sprintf(str+j, "\n");
        }
        j += sprintf(str+j, "0x%.2x, ", buf[i]);
    }
    sprintf(str+j, "\n");
    return 0;
}

CK_RV get_function_list(CK_CHAR *api_path,
    CK_FUNCTION_LIST_PTR *pfuncs)
{
    CK_RV rc = CKR_OK;
    CK_C_GetFunctionList pfoo = NULL;

    printf("get_function_list...\n");
#ifdef WIN32
    dll = LoadLibrary(api_path);
#else
    dll = dlopen(api_path, RTLD_NOW);
#endif
}
```

```
#endif
    if ( dll == NULL ) {
        printf("Can't load PKCS#11 API library.\n");
#ifdef WIN32
        printf("dlerror: %s\n", dlerror());
#endif
        return CKR_FUNCTION_FAILED;
    }
#ifdef WIN32
    pfoo = (CK_C_GetFunctionList)GetProcAddress(
        dll , "C_GetFunctionList");
#else
    pfoo = (CK_C_GetFunctionList)dlsym(
        dll , "C_GetFunctionList");
#endif
    if (pfoo == NULL ) {
        printf("Couldn't get C_GetFunctionList address\n");
        return CKR_FUNCTION_FAILED;
    }

    rc = pfoo(pfuncs);
    if (rc != CKR_OK) {
        show_error("Error in C_GetFunctionList", rc );
        return rc;
    }
#ifdef WIN32
    pctrl = (CK_C_Ctrl)GetProcAddress(dll , "C_Ctrl");
#else
    pctrl = (CK_C_Ctrl)dlsym(dll , "C_Ctrl");
#endif
    printf("Looks OK\n");
    return CKR_OK;
}

CK_RV init(CK_CHAR *api_path ,
           CK_FUNCTION_LIST_PTR *pfuncs ,
           CK_C_INITIALIZE_ARGS *pinit_args)
{
    CK_RV rc = CKR_OK;

    rc = get_function_list(api_path , pfuncs);
    if (rc != CKR_OK)
        return rc;
    rc = (*pfuncs)->C_Initialize(pinit_args);
}
```

```
    if (rc != CKR_OK) {
        show_error("C_Initialize", rc);
        cleanup(*pfuncs);
    }
    printf("C_Initialize OK\n");
    rc = CKR_OK;
    return rc;
}

CK_RV AppCreateMutex( void **pmutex )
{
#ifdef WIN32
    CRITICAL_SECTION *mutex;
#else
    pthread_mutex_t *mutex;
#endif
#ifdef WIN32
    mutex = (CRITICAL_SECTION *)malloc(sizeof(CRITICAL_SECTION));
    InitializeCriticalSection(mutex);
    *(CRITICAL_SECTION **)pmutex = mutex;
#else
    mutex = (pthread_mutex_t *)malloc(sizeof(pthread_mutex_t));
    pthread_mutex_init(mutex, NULL);
    *(pthread_mutex_t **)pmutex = mutex;
#endif
    return CKR_OK;
}

CK_RV AppDestroyMutex( void *mutex )
{
    if (mutex == NULL) {
        fprintf(stderr, "AppDestroyMutex: NULL mutex\n");
    } else {
#ifdef WIN32
        DeleteCriticalSection((CRITICAL_SECTION *)mutex);
        free(mutex);
#else
        pthread_mutex_destroy((pthread_mutex_t *)mutex);
        free(mutex);
#endif
    }
    return CKR_OK;
}
```



```
CK_RV AppLockMutex( void *mutex )
{
    if (mutex == NULL) {
        fprintf(stderr, "AppLockMutex: NULL mutex\n");
    } else {
#ifdef WIN32
        EnterCriticalSection(((CRITICAL_SECTION *)mutex));
#else
        pthread_mutex_lock((pthread_mutex_t *)mutex);
#endif
    }
    return CKR_OK;
}

CK_RV AppUnlockMutex( void *mutex )
{
    if (mutex == NULL) {
        fprintf(stderr, "AppUnlockMutex: NULL mutex\n");
    } else {
#ifdef WIN32
        LeaveCriticalSection(((CRITICAL_SECTION *)mutex));
#else
        pthread_mutex_unlock((pthread_mutex_t *)mutex);
#endif
    }
    return CKR_OK;
}

CK_RV cleanup(CK_FUNCTION_LIST_PTR funcs){
    CK_RV rc; // Return Code
    rc = funcs->C_Finalize(NULL);
    return rc;
}

CK_RV get_slot_list(CK_FUNCTION_LIST_PTR funcs ,
                    CK_SLOT_ID **ppSlotList ,
                    CK_ULONG *pulSlotCount)
{
    CK_RV rc;

    rc = funcs->C_GetSlotList(CK_FALSE, NULL, pulSlotCount);
    if (rc != CKR_OK) {
        *ppSlotList = NULL;
    }
}
```

```

        printf("Get slot list result: 0x%x\n", rc);
        return rc;
    }
    if (*pulSlotCount > 0)
    {
        *ppSlotList = (CK_SLOT_ID_PTR) malloc(
            *pulSlotCount * sizeof(CK_SLOT_ID));
        rc = funcs->C_GetSlotList(CK_FALSE, *ppSlotList,
            pulSlotCount);
        if (rc != CKR_OK) {
            *ppSlotList = NULL;
            printf("Get slot list result: 0x%x\n", rc);
            return rc;
        }
    } else {
        *ppSlotList = NULL;
        return CKR_OK;
    }
    printf("get_slot_list OK\n");
    return CKR_OK;
}

CK_RV find_slot_with_token(CK_FUNCTION_LIST_PTR funcs,
                           CK_SLOT_ID *pSlotList,
                           CK_ULONG ulSlotCount,
                           CK_SLOT_ID *pSlotId)
{
    CK_RV rc;
    CK_ULONG i;
    CK_SLOT_INFO sinfo;

    // Ищем слот с токеном
    for(i=0; i<ulSlotCount; ++i)
    {
        rc = funcs->C_GetSlotInfo(pSlotList[i], &sinfo);
        if (rc == CKR_OK)
        {
            if ((sinfo.flags & CKF_TOKEN_PRESENT) ==
                CKF_TOKEN_PRESENT)
            {
                // Хватаем первый попавшийся слот с токеном
                *pSlotId = pSlotList[i];
                rc = display_token_info(funcs, pSlotList[i]);
                return rc;
            }
        }
    }
}

```

```
    }  
    }  
    }  
    return CKR_FUNCTION_FAILED;  
}  
  
typedef struct {  
    CK_MECHANISM_TYPE type;  
    char *name;  
} MECH_INFO;  
  
static MECH_INFO mech [] = {  
    {CKM_GOSTR3410_KEY_PAIR_GEN,  
     "CKM_GOSTR3410_KEY_PAIR_GEN"},  
    {CKM_GOSTR3410_512_KEY_PAIR_GEN,  
     "CKM_GOSTR3410_512_KEY_PAIR_GEN"},  
    {CKM_GOSTR3410,  
     "CKM_GOSTR3410"},  
    {CKM_GOSTR3410_512,  
     "CKM_GOSTR3410_512"},  
    {CKM_GOSTR3410_WITH_GOSTR3411,  
     "CKM_GOSTR3410_WITH_GOSTR3411"},  
    {CKM_GOSTR3410_WITH_GOSTR3411_12_256,  
     "CKM_GOSTR3410_WITH_GOSTR3411_12_256"},  
    {CKM_GOSTR3410_WITH_GOSTR3411_12_512,  
     "CKM_GOSTR3410_WITH_GOSTR3411_12_512"},  
    {CKM_GOSTR3410_KEY_WRAP,  
     "CKM_GOSTR3410_KEY_WRAP"},  
    {CKM_GOSTR3410_DERIVE,  
     "CKM_GOSTR3410_DERIVE"},  
    {CKM_GOSTR3410_12_DERIVE,  
     "CKM_GOSTR3410_12_DERIVE"},  
    {CKM_GOSTR3411,  
     "CKM_GOSTR3411"},  
    {CKM_GOSTR3411_12_256,  
     "CKM_GOSTR3411_12_256"},  
    {CKM_GOSTR3411_12_512,  
     "CKM_GOSTR3411_12_512"},  
    {CKM_GOSTR3411_HMAC,  
     "CKM_GOSTR3411_HMAC"},  
    {CKM_GOSTR3411_12_256_HMAC,  
     "CKM_GOSTR3411_12_256_HMAC"},  
    {CKM_GOSTR3411_12_512_HMAC,  
     "CKM_GOSTR3411_12_512_HMAC"}  
};
```

```
{CKM_GOSTR3411_12_256_HMAC_KDF,  
  "CKM_GOSTR3411_12_256_HMAC_KDF"},  
{CKM_GOST28147_KEY_GEN,  
  "CKM_GOST28147_KEY_GEN"},  
{CKM_GOST28147_ECB,  
  "CKM_GOST28147_ECB"},  
{CKM_GOST28147,  
  "CKM_GOST28147"},  
{CKM_GOST28147_MAC,  
  "CKM_GOST28147_MAC"},  
{CKM_GOST28147_KEY_WRAP,  
  "CKM_GOST28147_KEY_WRAP"},  
{CKM_GOST28147_CNT,  
  "CKM_GOST28147_CNT"},  
{CKM_GOST28147_KEY_CPDIVERSIFY,  
  "CKM_GOST28147_KEY_CPDIVERSIFY"},  
{CKM_TLS_GOST_PRFB,  
  "CKM_TLS_GOST_PRFB"},  
{CKM_TLS_GOST_PRE_MASTER_KEY_GEN,  
  "CKM_TLS_GOST_PRE_MASTER_KEY_GEN"},  
{CKM_TLS_GOST_MASTER_KEY_DERIVE,  
  "CKM_TLS_GOST_MASTER_KEY_DERIVE"},  
{CKM_TLS_GOST_KEY_AND_MAC_DERIVE,  
  "CKM_TLS_GOST_KEY_AND_MAC_DERIVE"},  
{CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC,  
  "CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC"},  
{CKM_GOST28147_PKCS8_KEY_WRAP,  
  "CKM_GOST28147_PKCS8_KEY_WRAP"},  
{CKM_GOSTR3410_PUBLIC_KEY_DERIVE,  
  "CKM_GOSTR3410_PUBLIC_KEY_DERIVE"},  
{CKM_PKCS5_PBKDF2,  
  "CKM_PKCS5_PBKDF2"},  
};  
static int MECH_NUM = sizeof(mech)/sizeof(MECH_INFO);  
  
char *get_mechanism_name(CK_MECHANISM_TYPE mech_type) {  
  int i;  
  for (i=0; i<MECH_NUM; i++) {  
    if (mech[i].type == mech_type) {  
      return mech[i].name;  
    }  
  }  
  return NULL;  
}
```

```
CK_RV
display_pkcs11_info(CK_FUNCTION_LIST_PTR funcs){

    CK_RV rc;
    CK_INFO CryptokiInfo;

    memset(&CryptokiInfo, 0, sizeof(CK_INFO));
    rc = funcs->C_GetInfo(&CryptokiInfo);
    if (rc != CKR_OK) {
        show_error("C_GetInfo", rc);
        return rc;
    }

    /* display the header and information */
    printf("PKCS#11 Info\n");
    printf("\tVersion %d.%d \n", CryptokiInfo.cryptokiVersion.
        major,
        CryptokiInfo.cryptokiVersion.minor);
    printf("\tManufacturer: %.32s \n", CryptokiInfo.manufacturerID
    );
    printf("\tFlags: 0x%X \n", CryptokiInfo.flags);
    printf("\tLibrary Description: %.32s\n",
        CryptokiInfo.libraryDescription);
    printf("\tLibrary Version %d.%d \n",
        CryptokiInfo.libraryVersion.major,
        CryptokiInfo.libraryVersion.minor);

    return rc;
}

CK_RV
display_slot_info(CK_FUNCTION_LIST_PTR funcs, CK_SLOT_ID SlotId)
{
    CK_RV rc; // Return Code
    CK_SLOT_INFO SlotInfo; // Structure to hold slot information

    memset(&SlotInfo, 0, sizeof(CK_SLOT_INFO));
    rc = funcs->C_GetSlotInfo(SlotId, &SlotInfo);
    if (rc != CKR_OK) {
        printf("Error getting slot info: 0x%X\n", rc);
        return rc;
    }
    // Display the slot information
}
```

```
printf("Slot #%d Info\n", SlotId);
printf("\tDescription: %.64s\n", SlotInfo.slotDescription);
printf("\tManufacturer: %.32s\n", SlotInfo.manufacturerID);
printf("\tFlags: 0x%X\n", SlotInfo.flags);
printf("\tHardware Version: %d.%d\n",
SlotInfo.hardwareVersion.major,
SlotInfo.hardwareVersion.minor);
printf("\tFirmware Version: %d.%d\n",
SlotInfo.firmwareVersion.major,
SlotInfo.firmwareVersion.minor);

return CKR_OK;
}

CK_RV
display_token_info(CK_FUNCTION_LIST_PTR funcs, CK_SLOT_ID SlotId
) {
    CK_RV rc; // Return Code
    CK_TOKEN_INFO TokenInfo; // Variable to hold Token Information

    memset(&TokenInfo, 0, sizeof(CK_TOKEN_INFO));
    rc = funcs->C_GetTokenInfo(SlotId, &TokenInfo);
    if (rc != CKR_OK) {
        printf("Error getting token info: 0x%X\n", rc);
        return rc;
    }
    // Display the token information
    printf("Token #%d Info:\n", SlotId);
    printf("\tLabel: %.32s\n", TokenInfo.label);
    printf("\tManufacturer: %.32s\n", TokenInfo.manufacturerID);
    printf("\tModel: %.16s\n", TokenInfo.model);
    printf("\tSerial Number: %.16s\n", TokenInfo.serialNumber);
    printf("\tFlags: 0x%X\n", TokenInfo.flags);
    printf("\tSessions: %d/%d\n", TokenInfo.ulSessionCount,
TokenInfo.ulMaxSessionCount);
    printf("\tR/W Sessions: %d/%d\n",
TokenInfo.ulRwSessionCount, TokenInfo.ulMaxRwSessionCount);
    printf("\tPIN Length: %d-%d\n", TokenInfo.ulMinPinLen,
TokenInfo.ulMaxPinLen);
    printf("\tPublic Memory: 0x%X/0x%X\n",
TokenInfo.ulFreePublicMemory, TokenInfo.ulTotalPublicMemory);
    printf("\tPrivate Memory: 0x%X/0x%X\n",
TokenInfo.ulFreePrivateMemory, TokenInfo.ulTotalPrivateMemory)
;
}
```

```
printf("\tHardware Version: %d.%d\n",
    TokenInfo.hardwareVersion.major,
    TokenInfo.hardwareVersion.minor);
printf("\tFirmware Version: %d.%d\n",
    TokenInfo.firmwareVersion.major,
    TokenInfo.firmwareVersion.minor);
printf("\tTime: %.16s\n", TokenInfo.utcTime);

return CKR_OK;
}

CK_RV display_mechanisms(
    CK_FUNCTION_LIST_PTR funcs, CK_SLOT_ID slot_id)
{
    CK_RV rc;
    CK_MECHANISM_TYPE_PTR MechanismList = NULL;
    CK_MECHANISM_INFO MechanismInfo;
    CK_ULONG MechanismCount = 0;
    CK_ULONG i;
    char *mech_name = NULL;

    rc = funcs->C_GetMechanismList(slot_id, NULL_PTR,
        &MechanismCount);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_GetMechanismList: 0x%x\n", rc);
        return rc;
    }

    // Allocate enough memory to store all the supported
    // mechanisms
    MechanismList = (CK_MECHANISM_TYPE_PTR) malloc(MechanismCount
        *
        sizeof(CK_MECHANISM_TYPE));

    // This time get the mechanism list */
    rc = funcs->C_GetMechanismList(slot_id, MechanismList,
        &MechanismCount);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_GetMechanismList: 0x%x\n", rc);
        return rc;
    }

    // For each Mechanism in the List
```

```

for (i = 0; i < MechanismCount; i++){
    // Get the Mechanism Info and display it
    printf("Mechanism #%d\n", i);
    mech_name = get_mechanism_name(MechanismList[i]);
    if (mech_name) {
        printf("\tMechanism: 0x%X(%s)\n",
            MechanismList[i], mech_name);
    } else {
        printf("\tMechanism: 0x%X\n", MechanismList[i]);
    }
    rc = funcs->C_GetMechanismInfo(slot_id,
        MechanismList[i], &MechanismInfo);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_GetMechanismInfo: 0x%x\n", rc);
//        return rc;
    } else {
        printf("\tKey Size: %d-%d\n",
            MechanismInfo.ulMinKeySize,
            MechanismInfo.ulMaxKeySize);
        printf("\tFlags: 0x%X\n", MechanismInfo.flags);
    }
}

// Free the memory we allocated for the mechanism list
free (MechanismList);
return CKR_OK;
}

// Функция для консольного ввода PIN
CK_RV
get_pin(CK_CHAR ** pin){
    int size = PIN_SIZE, count = 0;
    char buff[PIN_SIZE] = { 0 }, c = 0;

#ifdef WIN32
    /* Turn off echoing to the terminal when getting the password
    */
    echo(FALSE);
#endif
    /* Get each character and print out a '*' for each input */
    for (count = 0; (c != LINE_FEED) &&
        (c != CARRIAGE_RETURN) && (count < PIN_SIZE); count++){
#ifdef WIN32
        buff[count] = getc(stdin);

```



```
#else
    buff[count] = _getch();
    if (buff[count] == END_OF_TEXT)
    {
        // Input terminated (Ctrl-C)
        memset(buff, 0, sizeof(buff));
        *pin = NULL;
        return CKR_CANCEL;
    }
#endif
c = buff[count];
if ((c != LINE_FEED) && (c != BACK_SPACE) &&
    (c != CARRIAGE_RETURN))
    printf("*");
if (c == BACK_SPACE) {
    printf("%c%c%c", BACK_SPACE, ' ', BACK_SPACE);
    count -= 2;
}
fflush(stdout);
}
#endif WIN32
echo(TRUE);
#endif

/* After we get the password go to the next line */
printf("\n");
fflush(stdout);

// Allocate 80 bytes for the user PIN
*pin = (unsigned char *) malloc(PIN_SIZE);
if (*pin == NULL) {
    memset(buff, 0, sizeof(buff));
    return CKR_HOST_MEMORY;
}

/* Strip the carriage return from
 * the user input (it is not part of the PIN)
 * and put the PIN in the return buffer */
buff[count-1] = '\0'; //NULL;
// keep the trailing null for the strlen
memcpy(*pin, buff, strlen(buff)+1);
memset(buff, 0, sizeof(buff));
return CKR_OK;
}
```

```
#ifndef WIN32
int
echo(int bbool){
    struct termios term;

    /* flush standard out to make sure everything
     * that needs to be displayed has
     * been displayed */
    fflush(stdout);

    /* get the current terminal attributes */
    if (tcgetattr(STDIN_FILENO, &term) != 0)
        return -1;

    /* Since we are calling this function
     * we must want to read in a char at a
     * time. Therefore set the cc structure
     * before setting the terminal attrs */
    term.c_cc[VMIN] = 1;
    term.c_cc[VTIME] = 0;

    /* If we are turning off the display
     * of input characters AND with the inverse
     * of the ECHO mask, if we are turning
     * on the display OR with the ECHO mask.
     * We also set if we are reading
     * in canonical or noncanonical mode. */
    if (bbool)
        term.c_lflag |= (ECHO | ICANON);
    else
        term.c_lflag &= ~(ECHO | ICANON);

    /* Set the attributes, and flush the streams
     * so that any input already
     * displayed on the terminal is invalid */
    if (tcsetattr(STDIN_FILENO, TCSAFLUSH, &term) != 0)
        return -1;

    return 0;
}
#endif

// Простенький колбэк для консольного ввода PIN.
```

```
// Эстеты могут написать здесь кроссплатформенное
// графическое приложение )
CK_RV pin_callback(
    CK_SESSION_HANDLE hSession,
    CK_NOTIFICATION event, // CKN_LISSI_PIN_CALLBACK
    CK_VOID_PTR pApplication // "Enter User/SO PIN please:"
)
{
    if (event == CKN_LISSI_PIN_CALLBACK) {
        CK_CHAR *pin_buf = NULL;
        CK_RV rc = CKR_OK;

        // Выдача приглашения ввести PIN
        printf((CK_CHAR *)pApplication);
        // get_pin сама выделяет память для pin_buf
        rc = get_pin(&pin_buf);
        if (rc == CKR_OK) {
            // Передаем значение PIN в библиотеку через pApplication
            memcpy(pApplication, pin_buf, strlen(pin_buf)+1);
            // Чистим буфер, выделенный get_pin
            memset(pin_buf, 0, strlen(pin_buf)+1);
            free(pin_buf);
            return CKR_OK;
        } else {
            // Ввод PIN был сброшен пользователем.
            // На всякий случай проверяем и чистим pin_buf,
            // хотя get_pin вроде-бы возвращает нулевой буфер
            // при возникновении ошибки.
            if (pin_buf) {
                memset(pin_buf, 0, strlen(pin_buf)+1);
                free(pin_buf);
            }
            return CKR_CANCEL;
        }
    } else {
        // По стандарту v2.30 нужно доброжелательно
        // игнорировать чуждые нам сообщения.
        return CKR_OK;
    }
}

CK_RV test_main(int argc, char *argv[])
{
    CK_RV rc = CKR_OK;
```

```
int i;

for (i=1; i<argc; i++) {
    if (strcmp("-api", argv[i]) == 0) {
        i++;
        api_path = argv[i];
    } else if (strcmp("-user_pin", argv[i]) == 0) {
        i++;
        user_pin = argv[i];
    }
}

rc = init(api_path, &funcs, NULL);
if (rc != CKR_OK) {
    fprintf(stderr, "ERROR calling init, rc = 0x%x\n", rc);
    return rc;
}
rc = get_slot_list(funcs,
                  &SlotList,
                  &SlotCount);
if (rc != CKR_OK) {
    printf("get_slot_list failed, rc = 0x%x\n", rc);
    return rc;
}
rc = find_slot_with_token(funcs,
                          SlotList,
                          SlotCount,
                          &SlotId);

if (rc != CKR_OK) {
    printf("find_slot_with_token failed, rc = 0x%x\n", rc);
    return rc;
}
return CKR_OK;
}
CK_RV check_value(CK_BYTE *value, CK_ULONG len,
                 CK_BYTE *eth, CK_ULONG eth_len)
{
    CK_RV rc = CKR_OK;

    if (len != eth_len) {
        printf("Invalid value length: %u != %u\n", len, eth_len);
        rc = CKR_DATA_LEN_RANGE;
    } else {
        if (memcmp(value, eth, eth_len) != 0) {
```

```
    printf("Invalid value:\n");
    print_hex(value, len);
    printf("Ethalon:\n");
    print_hex(eth, eth_len);
    rc = CKR_DATA_INVALID;
} else {
    printf("Test value OK\n");
}
}
return rc;
}
```

3.2 Инициализация библиотеки

После загрузки приложение должно вызвать функцию `C_Initialize`. В простейшем случае функция могла бы быть вызвана следующим образом:

```
funcs->C_Initialize(NULL);
```

Аргумент функции `C_Initialize` может быть задан и более сложным образом при использовании многопоточных приложений (см. тестовый пример `threadmkobj.c`).

3.3 Примеры программ

Исходные тексты тестов LS11SW могут служить примерами прикладных программ. Кроме того, ниже приводятся некоторые примеры программ, которые можно использовать в качестве образца при написании собственных программ. Приведенные здесь примеры демонстрируют далеко не все возможности LS11SW, поэтому рекомендуется ознакомиться с оригиналами тестовых примеров для различных механизмов в папке `tests`, а также с общими файлами `test_common.h` и `test_common.c` в папке `tests/common`.

По умолчанию, все тестовые программы работают с первым попавшимся токеном для библиотеки `ls11sw2012`. При этом предполагается, что PIN пользователя равен `01234567`. Если нужно запустить тестовый пример с другой библиотекой и/или с другим PIN, то это можно сделать в аргументах командной строки. Для библиотеки `ls11sw2012` это выглядит так:

```
>ckm_gostr3410_key_pair_gen -api ls11sw2012 -user_pin 01234567
```

3.3.1 Получение информации о слотах и токенах

Программа `info.c` выдает информацию о библиотеке, слотах и токенах.

Листинг 3.3: info.c

```
#include "test_common.h"

int
main(int argc, char *argv[]) {
    CK_RV rc;
    CK_SLOT_INFO sinfo;
    CK_ULONG i;

    printf("Info test\n");

    for (i=1; i<(CK_ULONG) argc; i++) {
        if (strcmp("-api", argv[i]) == 0) {
            ++i;
            api_path = argv[i];
        }
    }
    rc = init(api_path, &funcs, NULL);
    if (rc != CKR_OK) {
        printf("init failed\n");
        return rc;
    }

    rc = funcs->C_GetSlotList(CK_FALSE, NULL, &SlotCount);
    if (rc != CKR_OK) {
        SlotList = NULL;
        printf("Get slot list result: 0x%x\n", rc);
        return rc;
    }
    printf("%d slots found\n", SlotCount);
    if (SlotCount > 0)
    {
        SlotList = (CK_SLOT_ID_PTR) malloc(SlotCount * sizeof(
        CK_SLOT_ID));
        rc = funcs->C_GetSlotList(CK_FALSE, SlotList, &SlotCount);
        if (rc != CKR_OK) {
            SlotList = NULL;
            SlotCount = 0;
            printf("Get slot list result: 0x%x\n", rc);
            return rc;
        }
    }
    else {
        SlotList = NULL;
    }
}
```

```
    return CKR_OK;
}

printf("Cycle through slot list\n");
for (i = 0; i < SlotCount; i++)
{
    SlotId = SlotList[i];
    rc = display_slot_info(funcs, SlotId);
    if (rc != CKR_OK) {
        printf("display slot info failed\n");
        return rc;
    }
    rc = funcs->C_GetSlotInfo(SlotId, &sinfo);
    if (sinfo.flags & CKF_TOKEN_PRESENT) {
        rc = display_token_info(funcs, SlotId);
        if (rc != CKR_OK) {
            printf("display_token_info failed\n");
            return rc;
        }
    }
}

if (SlotList){
    free(SlotList);
    SlotList=NULL;
}

printf("Info test SUCCESS\n");
return CKR_OK;
}
```

3.3.2 Работа с объектами данных

Программа `sko_data.c` демонстрирует работу с объектами типа `CKO_DATA`.

Листинг 3.4: `sko_data.c`

```
#include "test_common.h"

int
main(int argc, char *argv[]) {
    CK_RV rc;
    CK_FLAGS flags = 0;
    CK_SESSION_HANDLE sess;
```

```

CK_ULONG i;
static CK_OBJECT_CLASS oclass_data = CKO_DATA;
static CK_BBOOL ltrue = CK_TRUE;
static CK_BBOOL lfalse = CK_FALSE;
CK_ATTRIBUTE data_find [] = {
    {CKA_CLASS, &oclass_data, sizeof(oclass_data)},
};
CK_UTF8CHAR label [] = "data";
CK_CHAR app [] = "ls_hw11_library";
CK_BYTE data_value [] = {
    0x66, 0x6F, 0x40, 0x63, 0x72, 0x79, 0x70, 0x74,
    0x6F, 0x70, 0x72, 0x6F, 0x2E, 0x72, 0x75, 0x31,
    0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06,
    0x13, 0x02, 0x52, 0x55, 0x31, 0x13, 0x30, 0x11,
};
CK_ATTRIBUTE data_create [] = {
    {CKA_CLASS, &oclass_data, sizeof(oclass_data)},
    {CKA_APPLICATION, app, strlen(app)+1},
    {CKA_TOKEN, &ltrue, sizeof(ltrue)},
    {CKA_PRIVATE, &lfalse, sizeof(lfalse)},
    {CKA_LABEL, label, strlen(label)+1},
    {CKA_VALUE, data_value, sizeof(data_value)},
};
CK_BYTE get_value [1024];
CK_ATTRIBUTE tmpl_value [] = {
    {CKA_VALUE, get_value, sizeof(get_value)},
};
CK_ATTRIBUTE data_value_find [] = {
    {CKA_CLASS, &oclass_data, sizeof(oclass_data)},
    {CKA_VALUE, data_value, sizeof(data_value)},
};
CK_ULONG count, number;
CK_OBJECT_HANDLE hObj = CK_INVALID_HANDLE;
CK_OBJECT_HANDLE hNewObj = CK_INVALID_HANDLE;

printf("CKO_DATA test\n");
for (i=1; i<(CK_ULONG)argc; i++) {
    if (strcmp("-api", argv[i]) == 0) {
        ++i;
        api_path = argv[i];
    } else if (strcmp("-user_pin", argv[i]) == 0) {
        ++i;
        user_pin = argv[i];
    }
}

```



```
}
rc = init(api_path, &funcs, NULL);
if (rc != CKR_OK) {
    printf("init failed\n");
    return rc;
}
rc = get_slot_list(funcs,
                  &SlotList,
                  &SlotCount);
if (rc != CKR_OK) {
    printf("get_slot_list failed, rc = 0x%x\n", rc );
    return rc;
}
rc = find_slot_with_token(funcs,
                          SlotList,
                          SlotCount,
                          &SlotId);

if (rc != CKR_OK) {
    printf("find_slot_with_token failed, rc = 0x%x\n", rc );
    return rc;
}

flags = CKF_SERIAL_SESSION | CKF_RW_SESSION;
rc = funcs->C_OpenSession( SlotId, flags, NULL, NULL, &sess );
if (rc != CKR_OK) {
    printf("C_OpenSession failed, rc = 0x%x\n", rc );
    return rc;
}

rc = funcs->C_Login(sess, CKU_USER, user_pin, strlen(user_pin)
);
if (rc != CKR_OK) {
    printf("C_Login failed, rc = 0x%x\n", rc );
    return rc;
}
printf("C_Login (CKU_USER) success\n");

// Ищем все доступные объекты CKO_DATA
rc = funcs->C_FindObjectsInit(sess,
                             data_find, sizeof(data_find)/sizeof(CK_ATTRIBUTE));
if (rc != CKR_OK) {
    printf("C_FindObjectsFinal failed, rc = 0x%x\n", rc );
    return rc;
}
}
```

```
count = 0;
do {
    number = 0;
    hObj = CK_INVALID_HANDLE;
    rc = funcs->C_FindObjects(sess, &hObj, 1, &number);
    if (rc != CKR_OK) {
        printf("C_FindObjects failed, rc = 0x%x\n", rc);
        return rc;
    }
    if (number > 0 && hObj != CK_INVALID_HANDLE) count++;
} while (number > 0 && hObj != CK_INVALID_HANDLE);
rc = funcs->C_FindObjectsFinal(sess);
if (rc != CKR_OK) {
    printf("C_FindObjectsFinal failed, rc = 0x%x\n", rc);
    return rc;
}
printf("%ld data objects found for session\n", count);

rc = funcs->C_CreateObject(sess,
    data_create, sizeof(data_create)/sizeof(CK_ATTRIBUTE),
    &hNewObj);
if (rc != CKR_OK) {
    printf("C_CreateObject failed, rc = 0x%x\n", rc);
    return rc;
}
printf("C_CreateObject success\n");

rc = funcs->C_GetAttributeValue(sess, hNewObj,
    tmpl_value, sizeof(tmpl_value)/sizeof(CK_ATTRIBUTE));
if (rc != CKR_OK) {
    printf("C_GetAttributeValue failed, rc = 0x%x\n", rc);
    return rc;
}
if (tmpl_value[0].ulValueLen != sizeof(data_value)) {
    printf("Invalid CKA_VALUE length\n");
    return -1;
}
if (memcmp(get_value, data_value, sizeof(data_value)) != 0) {
    printf("Invalid CKA_VALUE\n");
    return -1;
}
printf("C_GetAttributeValue success\n");

// Ищем все доступные объекты CKO_DATA по значению CKA_VALUE
```

```
rc = funcs->C_FindObjectsInit(sess ,
    data_value_find ,
    sizeof(data_value_find)/sizeof(CK_ATTRIBUTE));
if (rc != CKR_OK) {
    printf("C_FindObjectsFinal failed, rc = 0x%x\n", rc );
    return rc;
}
count = 0;
do {
    number = 0;
    hObj = CK_INVALID_HANDLE;
    rc = funcs->C_FindObjects(sess , &hObj, 1, &number);
    if (rc != CKR_OK) {
        printf("C_FindObjects failed, rc = 0x%x\n", rc );
        return rc;
    }
    if (number > 0 && hObj != CK_INVALID_HANDLE) count ++;
} while (number > 0 && hObj != CK_INVALID_HANDLE);
rc = funcs->C_FindObjectsFinal(sess);
if (rc != CKR_OK) {
    printf("C_FindObjectsFinal failed, rc = 0x%x\n", rc );
    return rc;
}
printf("%ld data objects found for given CKA_VALUE\n",
    count);

rc = funcs->C_DestroyObject(sess , hNewObj);
if (rc != CKR_OK) {
    printf("C_DestroyObject failed, rc = 0x%x\n", rc );
    return rc;
}
printf("C_DestroyObject success\n");

rc = funcs->C_CloseSession(sess);
if (rc != CKR_OK) {
    printf("C_CloseSession failed, rc = 0x%x\n", rc );
    return rc;
}

printf("CKO_DATA test SUCCESS\n");
return CKR_OK;
}
```

3.3.3 Работа с сертификатами

Программа `sko_certificate.c` демонстрирует работу с объектами типа `CKO_CERTIFICATE`.

Листинг 3.5: `sko_certificate.c`

```
#include "test_common.h"

int
main(int argc, char *argv[]) {
    CK_RV rc;
    CK_FLAGS flags = 0;
    CK_SESSION_HANDLE sess;
    CK_ULONG i;
    static CK_OBJECT_CLASS oclass_cert = CKO_CERTIFICATE;
    static CK_CERTIFICATE_TYPE ocert_type = CKC_X_509;
    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    CK_ATTRIBUTE cert_find[] = {
        {CKA_CLASS, &oclass_cert, sizeof(oclass_cert)},
    };
    CK_UTF8CHAR label[] = "cpcacert";
    CK_CHAR subject[] = "CryptoPro CA";
    // Размер сертификата - 583 байта
    CK_BYTE der_value[] = {
        0x30, 0x82, 0x02, 0x43, 0x30, 0x82, 0x01, 0xF0,
        0xA0, 0x03, 0x02, 0x01, 0x02, 0x02, 0x10, 0x69,
        0x84, 0x03, 0x28, 0x6A, 0xA6, 0x59, 0xBA, 0x46,
        0x35, 0x62, 0x2D, 0x49, 0xDE, 0x5F, 0xD3, 0x30,
        0x0A, 0x06, 0x06, 0x2A, 0x85, 0x03, 0x02, 0x02,
        0x03, 0x05, 0x00, 0x30, 0x65, 0x31, 0x20, 0x30,
        0x1E, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7,
        0x0D, 0x01, 0x09, 0x01, 0x16, 0x11, 0x69, 0x6E,
        0x66, 0x6F, 0x40, 0x63, 0x72, 0x79, 0x70, 0x74,
        0x6F, 0x70, 0x72, 0x6F, 0x2E, 0x72, 0x75, 0x31,
        0x0B, 0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06,
        0x13, 0x02, 0x52, 0x55, 0x31, 0x13, 0x30, 0x11,
        0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x0A, 0x43,
        0x52, 0x59, 0x50, 0x54, 0x4F, 0x2D, 0x50, 0x52,
        0x4F, 0x31, 0x1F, 0x30, 0x1D, 0x06, 0x03, 0x55,
        0x04, 0x03, 0x13, 0x16, 0x54, 0x65, 0x73, 0x74,
        0x20, 0x43, 0x65, 0x6E, 0x74, 0x65, 0x72, 0x20,
        0x43, 0x52, 0x59, 0x50, 0x54, 0x4F, 0x2D, 0x50,
        0x52, 0x4F, 0x30, 0x1E, 0x17, 0x0D, 0x30, 0x39,
        0x30, 0x34, 0x30, 0x37, 0x31, 0x32, 0x30, 0x32,
```

```
0x31, 0x35, 0x5A, 0x17, 0x0D, 0x31, 0x34, 0x31,
0x30, 0x30, 0x34, 0x30, 0x37, 0x30, 0x39, 0x34,
0x31, 0x5A, 0x30, 0x65, 0x31, 0x20, 0x30, 0x1E,
0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D,
0x01, 0x09, 0x01, 0x16, 0x11, 0x69, 0x6E, 0x66,
0x6F, 0x40, 0x63, 0x72, 0x79, 0x70, 0x74, 0x6F,
0x70, 0x72, 0x6F, 0x2E, 0x72, 0x75, 0x31, 0x0B,
0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13,
0x02, 0x52, 0x55, 0x31, 0x13, 0x30, 0x11, 0x06,
0x03, 0x55, 0x04, 0x0A, 0x13, 0x0A, 0x43, 0x52,
0x59, 0x50, 0x54, 0x4F, 0x2D, 0x50, 0x52, 0x4F,
0x31, 0x1F, 0x30, 0x1D, 0x06, 0x03, 0x55, 0x04,
0x03, 0x13, 0x16, 0x54, 0x65, 0x73, 0x74, 0x20,
0x43, 0x65, 0x6E, 0x74, 0x65, 0x72, 0x20, 0x43,
0x52, 0x59, 0x50, 0x54, 0x4F, 0x2D, 0x50, 0x52,
0x4F, 0x30, 0x63, 0x30, 0x1C, 0x06, 0x06, 0x2A,
0x85, 0x03, 0x02, 0x02, 0x13, 0x30, 0x12, 0x06,
0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01,
0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x1E,
0x01, 0x03, 0x43, 0x00, 0x04, 0x40, 0x02, 0xE4,
0xFF, 0xD1, 0xA6, 0xF6, 0x9C, 0x80, 0x9A, 0xDA,
0xEC, 0x7F, 0x4A, 0x78, 0xC1, 0xCC, 0x2D, 0xD3,
0xE5, 0x96, 0xEA, 0xCB, 0xED, 0x22, 0x32, 0x79,
0xB2, 0x02, 0xE2, 0xC6, 0x7C, 0x35, 0xE6, 0x74,
0x64, 0x1B, 0x09, 0x77, 0x11, 0x8C, 0x67, 0x3F,
0x0F, 0xD0, 0xE8, 0x23, 0xA6, 0x7E, 0x6D, 0x3B,
0x7F, 0xC4, 0xC4, 0x28, 0xFD, 0x2B, 0x1C, 0x68,
0x01, 0x20, 0xA0, 0x5C, 0xD8, 0x79, 0xA3, 0x78,
0x30, 0x76, 0x30, 0x0B, 0x06, 0x03, 0x55, 0x1D,
0x0F, 0x04, 0x04, 0x03, 0x02, 0x01, 0xC6, 0x30,
0x0F, 0x06, 0x03, 0x55, 0x1D, 0x13, 0x01, 0x01,
0xFF, 0x04, 0x05, 0x30, 0x03, 0x01, 0x01, 0xFF,
0x30, 0x1D, 0x06, 0x03, 0x55, 0x1D, 0x0E, 0x04,
0x16, 0x04, 0x14, 0x6D, 0x8F, 0x5E, 0x05, 0xD9,
0x5F, 0xAC, 0x91, 0x17, 0x94, 0x1E, 0x95, 0x9A,
0x05, 0x30, 0x38, 0x37, 0x7A, 0x10, 0x2A, 0x30,
0x12, 0x06, 0x09, 0x2B, 0x06, 0x01, 0x04, 0x01,
0x82, 0x37, 0x15, 0x01, 0x04, 0x05, 0x02, 0x03,
0x02, 0x00, 0x02, 0x30, 0x23, 0x06, 0x09, 0x2B,
0x06, 0x01, 0x04, 0x01, 0x82, 0x37, 0x15, 0x02,
0x04, 0x16, 0x04, 0x14, 0x7A, 0xC9, 0xC7, 0x09,
0xDB, 0x20, 0x1C, 0x96, 0x94, 0x2F, 0xFC, 0x46,
0xAD, 0x6D, 0x93, 0xD0, 0x5E, 0x69, 0x12, 0x0E,
0x30, 0x0A, 0x06, 0x06, 0x2A, 0x85, 0x03, 0x02,
```

```

0x02, 0x03, 0x05, 0x00, 0x03, 0x41, 0x00, 0x58,
0x73, 0xD2, 0x93, 0xBC, 0x63, 0x21, 0xB1, 0x0E,
0x73, 0x72, 0xEE, 0xF1, 0x72, 0xB5, 0x1B, 0x8B,
0xBB, 0xC9, 0x3B, 0x08, 0xBB, 0x4C, 0x5A, 0xF2,
0xE1, 0xA5, 0x35, 0x4F, 0x99, 0xC4, 0xD5, 0x52,
0x52, 0x70, 0x26, 0xDD, 0xAE, 0xD0, 0xA9, 0x27,
0xE9, 0xB6, 0x5B, 0x7D, 0x6F, 0x44, 0xFD, 0x26,
0x4D, 0xFD, 0xA1, 0x63, 0x74, 0x5C, 0x74, 0xD8,
0x49, 0x73, 0x0A, 0x77, 0x77, 0x63, 0x4D,

};
CK_ATTRIBUTE cert_create [] = {
    {CKA_CLASS, &oclass_cert, sizeof(oclass_cert)},
    {CKA_CERTIFICATE_TYPE, &ocert_type, sizeof(ocert_type)},
    {CKA_TOKEN, &ltrue, sizeof(ltrue)},
    {CKA_PRIVATE, &lfalse, sizeof(lfalse)},
    {CKA_SUBJECT, subject, strlen(subject)+1},
    {CKA_LABEL, label, strlen(label)+1},
    {CKA_VALUE, der_value, sizeof(der_value)},
};
CK_BYTE get_value[4096];
CK_ATTRIBUTE tmp_value [] = {
    {CKA_VALUE, get_value, sizeof(get_value)},
};
CK_ULONG count, number;
CK_OBJECT_HANDLE hObj = CK_INVALID_HANDLE;

printf("CKO_CERTIFICATE test\n");

for (i=1; i<(CK_ULONG) argc; i++) {
    if (strcmp("-api", argv[i]) == 0) {
        ++i;
        api_path = argv[i];
    } else if (strcmp("-user_pin", argv[i]) == 0) {
        ++i;
        user_pin = argv[i];
    }
}
rc = init(api_path, &funcs, NULL);
if (rc != CKR_OK) {
    printf("init failed\n");
    return rc;
}
rc = get_slot_list(funcs,

```

```
        &SlotList ,
        &SlotCount);
if (rc != CKR_OK) {
    printf("get_slot_list failed, rc = 0x%x\n", rc );
    return rc;
}
rc = find_slot_with_token(funcs ,
                          SlotList ,
                          SlotCount ,
                          &SlotId);

if (rc != CKR_OK) {
    printf("find_slot_with_token failed, rc = 0x%x\n", rc );
    return rc;
}

flags = CKF_SERIAL_SESSION | CKF_RW_SESSION;
rc = funcs->C_OpenSession( SlotId , flags , NULL , NULL , &sess );
if (rc != CKR_OK) {
    printf("C_OpenSession failed, rc = 0x%x\n", rc );
    return rc;
}

rc = funcs->C_Login(sess , CKU_USER, user_pin , strlen(user_pin)
);
if (rc != CKR_OK) {
    printf("C_Login failed, rc = 0x%x\n", rc );
    return rc;
}
printf("C_Login (CKU_USER) success\n");

// Ищем все доступные сертификаты
rc = funcs->C_FindObjectsInit(sess ,
    cert_find , sizeof(cert_find)/sizeof(CK_ATTRIBUTE));
if (rc != CKR_OK) {
    printf("C_FindObjectsFinal failed, rc = 0x%x\n", rc );
    return rc;
}
count = 0;
do {
    number = 0;
    hObj = CK_INVALID_HANDLE;
    rc = funcs->C_FindObjects(sess , &hObj , 1 , &number);
    if (rc != CKR_OK) {
        printf("C_FindObjects failed, rc = 0x%x\n", rc );
    }
}
```

```
    return rc;
}
if (number > 0 && hObj != CK_INVALID_HANDLE) count ++;
} while (number > 0 && hObj != CK_INVALID_HANDLE);
rc = funcs->C_FindObjectsFinal(sess);
if (rc != CKR_OK) {
    printf("C_FindObjectsFinal failed, rc = 0x%x\n", rc );
    return rc;
}
printf("%ld certificates found for session\n", count);

rc = funcs->C_CreateObject(sess,
    cert_create, sizeof(cert_create)/sizeof(CK_ATTRIBUTE), &hObj
);
if (rc != CKR_OK) {
    printf("C_CreateObject failed, rc = 0x%x\n", rc );
    return rc;
}
printf("C_CreateObject success\n");

rc = funcs->C_GetAttributeValue(sess, hObj,
    tpl_value, sizeof(tpl_value)/sizeof(CK_ATTRIBUTE));
if (rc != CKR_OK) {
    printf("C_GetAttributeValue failed, rc = 0x%x\n", rc );
    return rc;
}
if (tpl_value[0].ulValueLen != sizeof(der_value)) {
    printf("Invalid CKA_VALUE length\n");
    return -1;
}
if (memcmp(get_value, der_value, sizeof(der_value)) != 0 ) {
    printf("Invalid CKA_VALUE\n");
    return -1;
}
printf("C_GetAttributeValue success\n");

rc = funcs->C_DestroyObject(sess, hObj);
if (rc != CKR_OK) {
    printf("C_DestroyObject failed, rc = 0x%x\n", rc );
    return rc;
}

rc = funcs->C_CloseSession(sess);
if (rc != CKR_OK) {
```



```
    printf("C_CloseSession failed, rc = 0x%x\n", rc );
    return rc;
}

printf("CKO_CERTIFICATE test SUCCESS\n");
return CKR_OK;
}
```

3.3.4 Генерация дайджеста

ГОСТ Р34.11-94

Программа вычисляет хеш сообщения по алгоритму ГОСТ Р 34.11-94 и сравнивает результат с заранее заданным эталоном. Программа использует умалчиваемую конфигурацию, предполагая, что умалчиваемый программный токен уже готов к работе.

Заметим, что размер состояния контекста хеширования зависит от реализации и в разных версиях библиотеки может оказаться различным, поэтому не следует надеяться на фиксированный размер буфера при сохранении этого состояния с помощью функции `C_GetOperationState`.

Листинг 3.6: `ckm_gostr3411.c`

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost3411(CK_SESSION_HANDLE hSession);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
```

```
{
    CK_RV rc , ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession( SlotId ,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr ,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr , "C_OpenSession success\n");
    rc = funcs->C_GetMechanismInfo( SlotId , CKM_GOSTR3411, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr ,
            "\n==== Mechanism CKM_GOSTR3411 not supported =====\n"
        );
    } else {
        fprintf(stderr ,
            "\n===== CKM_GOSTR3411 test =====\n"
        );
        rc = test_gost3411(hSession);
        if (rc != CKR_OK) {
            fprintf(stderr ,
                "ERROR CKM_GOSTR3411 failed, rc = 0x%x\n", rc);
        } else {
            fprintf(stderr , "CKM_GOSTR3411 test passed.\n");
        }
    }
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr ,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr , "C_CloseSession success\n");
}

out:
    return rc;
}
```

```

CK_RV test_gost3411(CK_SESSION_HANDLE hSession)
{
    CK_RV rc;
    CK_BYTE value[256];
    CK_BYTE *state;
    CK_ULONG len, state_len;
    CK_OBJECT_HANDLE paramh = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc = {CKM_GOSTR3411, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    // CryptoPro gost3411 Test Param Set
    static CK_BYTE oid [] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x00
    };
    static CK_BYTE data1 [] =
        "Suppose the original message has length = 50 bytes";
    static CK_BYTE et1 [] = {
        0x47, 0x1a, 0xba, 0x57, 0xa6, 0x0a, 0x77, 0x0d,
        0x3a, 0x76, 0x13, 0x06, 0x35, 0xc1, 0xfb, 0xea,
        0x4e, 0xf1, 0x4d, 0xe5, 0x1f, 0x78, 0xb4, 0xae,
        0x57, 0xdd, 0x89, 0x3b, 0x62, 0xf5, 0x52, 0x08
    };
    static CK_BYTE data2 [] = {
        0xc3, 0x73, 0x0c, 0x5c, 0xbc, 0xca, 0xcf, 0x91,
        0x5a, 0xc2, 0x92, 0x67, 0x6f, 0x21, 0xe8, 0xbd,
        0x4e, 0xf7, 0x53, 0x31, 0xd9, 0x40, 0x5e, 0x5f,
        0x1a, 0x61, 0xdc, 0x31, 0x30, 0xa6, 0x50, 0x11
    };
    static CK_BYTE et2 [] = {
        0x38, 0x65, 0x45, 0xc7, 0x71, 0x4f, 0x6d, 0x0b,
        0xf1, 0x27, 0xad, 0x57, 0xca, 0xc0, 0x92, 0x0a,
        0xa3, 0xd8, 0x84, 0x62, 0xfa, 0x8a, 0x0e, 0x1d,
        0x30, 0xd5, 0xcd, 0x5a, 0x85, 0x84, 0xaf, 0xf1
    };

    static CK_BYTE data3 [] = {
        0x30, 0x82, 0x03, 0x9c, 0xa0, 0x03, 0x02, 0x01,
        0x02, 0x02, 0x10, 0x3a, 0xc3, 0xb8, 0xac, 0xec,
        0xfb, 0xd7, 0xae, 0x28, 0xb5, 0x92, 0x9f, 0xd2,
        0xec, 0x4c, 0xf3, 0x30, 0x08, 0x06, 0x06, 0x2a,
        0x85, 0x03, 0x02, 0x02, 0x03, 0x30, 0x81, 0xec,
        0x31, 0x19, 0x30, 0x17, 0x06, 0x09, 0x2a, 0x86,
        0x48, 0x86, 0xf7, 0x0d, 0x01, 0x09, 0x01, 0x16,
    };
}

```

```
0x0a,0x67,0x64,0x75,0x63,0x40,0x63,0x61,
0x2e,0x72,0x75,0x31,0x0b,0x30,0x09,0x06,
0x03,0x55,0x04,0x06,0x13,0x02,0x52,0x55,
0x31,0x15,0x30,0x13,0x06,0x03,0x55,0x04,
0x07,0x0c,0x0c,0xd0,0x9c,0xd0,0xbe,0xd1,
0x81,0xd0,0xba,0xd0,0xb2,0xd0,0xb0,0x31,
0x54,0x30,0x52,0x06,0x03,0x55,0x04,0x0a,
0x0c,0x4b,0xd0,0x93,0xd0,0xbb,0xd0,0xb0,
0xd0,0xb2,0xd0,0xbd,0xd1,0x8b,0xd0,0xb9,
0x20,0xd0,0x94,0xd0,0xbe,0xd0,0xb2,0xd0,
0xb5,0xd1,0x80,0xd0,0xb5,0xd0,0xbd,0xd0,
0xbd,0xd1,0x8b,0xd0,0xb9,0x20,0xd0,0xa3,
0xd0,0xb4,0xd0,0xbe,0xd1,0x81,0xd1,0x82,
0xd0,0xbe,0xd0,0xb2,0xd0,0xb5,0xd1,0x80,
0xd1,0x8f,0xd1,0x8e,0xd1,0x89,0xd0,0xb8,
0xd0,0xb9,0x20,0xd0,0xa6,0xd0,0xb5,0xd0,
0xbd,0xd1,0x82,0xd1,0x80,0x31,0x2c,0x30,
0x2a,0x06,0x03,0x55,0x04,0x0b,0x0c,0x23,
0xd0,0xa6,0xd0,0xb5,0xd0,0xbd,0xd1,0x82,
0xd1,0x80,0x20,0xd0,0xa1,0xd0,0xb5,0xd1,
0x80,0xd1,0x82,0xd0,0xb8,0xd1,0x84,0xd0,
0xb8,0xd0,0xba,0xd0,0xb0,0xd1,0x86,0xd0,
0xb8,0xd0,0xb8,0x31,0x27,0x30,0x25,0x06,
0x03,0x55,0x04,0x03,0x0c,0x1e,0xd0,0x93,
0xd0,0x94,0xd0,0xa3,0xd0,0xa6,0x20,0xd0,
0xa1,0xd1,0x82,0xd0,0xb0,0xd0,0xbd,0xd0,
0xb4,0xd0,0xb0,0xd1,0x80,0xd1,0x82,0x20,
0xd0,0xa3,0xd0,0xa6,0x30,0x1e,0x17,0x0d,
0x30,0x34,0x30,0x31,0x30,0x39,0x31,0x32,
0x33,0x33,0x32,0x39,0x5a,0x17,0x0d,0x31,
0x34,0x30,0x31,0x30,0x36,0x31,0x32,0x33,
0x33,0x32,0x39,0x5a,0x30,0x81,0xec,0x31,
0x19,0x30,0x17,0x06,0x09,0x2a,0x86,0x48,
0x86,0xf7,0x0d,0x01,0x09,0x01,0x16,0x0a,
0x67,0x64,0x75,0x63,0x40,0x63,0x61,0x2e,
0x72,0x75,0x31,0x0b,0x30,0x09,0x06,0x03,
0x55,0x04,0x06,0x13,0x02,0x52,0x55,0x31,
0x15,0x30,0x13,0x06,0x03,0x55,0x04,0x07,
0x0c,0x0c,0xd0,0x9c,0xd0,0xbe,0xd1,0x81,
0xd0,0xba,0xd0,0xb2,0xd0,0xb0,0x31,0x54,
0x30,0x52,0x06,0x03,0x55,0x04,0x0a,0x0c,
0x4b,0xd0,0x93,0xd0,0xbb,0xd0,0xb0,0xd0,
0xb2,0xd0,0xbd,0xd1,0x8b,0xd0,0xb9,0x20,
0xd0,0x94,0xd0,0xbe,0xd0,0xb2,0xd0,0xb5,
```

```
0xd1,0x80,0xd0,0xb5,0xd0,0xbd,0xd0,0xbd,
0xd1,0x8b,0xd0,0xb9,0x20,0xd0,0xa3,0xd0,
0xb4,0xd0,0xbe,0xd1,0x81,0xd1,0x82,0xd0,
0xbe,0xd0,0xb2,0xd0,0xb5,0xd1,0x80,0xd1,
0x8f,0xd1,0x8e,0xd1,0x89,0xd0,0xb8,0xd0,
0xb9,0x20,0xd0,0xa6,0xd0,0xb5,0xd0,0xbd,
0xd1,0x82,0xd1,0x80,0x31,0x2c,0x30,0x2a,
0x06,0x03,0x55,0x04,0x0b,0x0c,0x23,0xd0,
0xa6,0xd0,0xb5,0xd0,0xbd,0xd1,0x82,0xd1,
0x80,0x20,0xd0,0xa1,0xd0,0xb5,0xd1,0x80,
0xd1,0x82,0xd0,0xb8,0xd1,0x84,0xd0,0xb8,
0xd0,0xba,0xd0,0xb0,0xd1,0x86,0xd0,0xb8,
0xd0,0xb8,0x31,0x27,0x30,0x25,0x06,0x03,
0x55,0x04,0x03,0x0c,0x1e,0xd0,0x93,0xd0,
0x94,0xd0,0xa3,0xd0,0xa6,0x20,0xd0,0xa1,
0xd1,0x82,0xd0,0xb0,0xd0,0xbd,0xd0,0xb4,
0xd0,0xb0,0xd1,0x80,0xd1,0x82,0x20,0xd0,
0xa3,0xd0,0xa6,0x30,0x63,0x30,0x1c,0x06,
0x06,0x2a,0x85,0x03,0x02,0x02,0x13,0x30,
0x12,0x06,0x07,0x2a,0x85,0x03,0x02,0x02,
0x23,0x01,0x06,0x07,0x2a,0x85,0x03,0x02,
0x02,0x1e,0x01,0x03,0x43,0x00,0x04,0x40,
0x50,0xab,0x7f,0xc4,0xcc,0x3d,0xd0,0xe2,
0xdd,0x86,0xda,0x19,0x6b,0x14,0x8c,0x78,
0xd9,0xca,0x58,0x67,0x62,0xf3,0xb7,0xba,
0x7b,0x2a,0xda,0xc1,0x9c,0x3f,0x87,0xeb,
0xf1,0xdc,0xaf,0x35,0xad,0x2d,0xe1,0xca,
0xed,0xc1,0x8b,0x82,0xde,0xa0,0x8b,0x95,
0xdd,0xa2,0xac,0x46,0x6a,0x8e,0xce,0x5d,
0x5a,0x16,0xba,0x03,0x29,0x72,0x38,0x27,
0xa3,0x82,0x01,0x14,0x30,0x82,0x01,0x10,
0x30,0x5a,0x06,0x03,0x55,0x1d,0x11,0x01,
0x01,0xff,0x04,0x50,0x30,0x4e,0x81,0x0b,
0x67,0x64,0x75,0x63,0x63,0x40,0x75,0x63,
0x2e,0x72,0x75,0xa4,0x3f,0x30,0x3d,0x31,
0x3b,0x30,0x39,0x06,0x03,0x55,0x04,0x03,
0x0c,0x32,0xd0,0x9f,0xd0,0xb5,0xd1,0x80,
0xd0,0xb2,0xd0,0xbe,0xd0,0xb5,0x20,0xd0,
0xa3,0xd0,0xbf,0xd0,0xbe,0xd0,0xbb,0xd0,
0xbd,0xd0,0xbe,0xd0,0xbc,0xd0,0xbe,0xd1,
0x87,0xd0,0xb5,0xd0,0xbd,0xd0,0xbd,0xd0,
0xbe,0xd0,0xb5,0x20,0xd0,0x9b,0xd0,0xb8,
0xd1,0x86,0xd0,0xbe,0x30,0x0f,0x06,0x03,
0x55,0x1d,0x0f,0x01,0x01,0xff,0x04,0x05,
```

```

0x03, 0x03, 0x07, 0xc6, 0x00, 0x30, 0x0f, 0x06,
0x03, 0x55, 0x1d, 0x13, 0x01, 0x01, 0xff, 0x04,
0x05, 0x30, 0x03, 0x01, 0x01, 0xff, 0x30, 0x1d,
0x06, 0x03, 0x55, 0x1d, 0x0e, 0x04, 0x16, 0x04,
0x14, 0xb1, 0x6e, 0x0e, 0xa4, 0x40, 0xbc, 0xf0,
0xd9, 0xb6, 0xf7, 0xef, 0xfa, 0xf0, 0x3d, 0xa1,
0x0c, 0xd2, 0x8f, 0xf1, 0xb6, 0x30, 0x71, 0x06,
0x03, 0x55, 0x1d, 0x1f, 0x04, 0x6a, 0x30, 0x68,
0x30, 0x66, 0xa0, 0x64, 0xa0, 0x62, 0x86, 0x60,
0x6c, 0x64, 0x61, 0x70, 0x3a, 0x2f, 0x2f, 0x31,
0x39, 0x32, 0x2e, 0x31, 0x36, 0x38, 0x2e, 0x36,
0x38, 0x2e, 0x37, 0x30, 0x2f, 0x6f, 0x3d, 0x72,
0x6f, 0x6f, 0x74, 0x2c, 0x63, 0x3d, 0x72, 0x75,
0x3f, 0x63, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69,
0x63, 0x61, 0x74, 0x65, 0x52, 0x65, 0x76, 0x6f,
0x63, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x4c, 0x69,
0x73, 0x74, 0x3f, 0x62, 0x61, 0x73, 0x65, 0x3f,
0x6f, 0x62, 0x6a, 0x65, 0x63, 0x74, 0x63, 0x6c,
0x61, 0x73, 0x73, 0x3d, 0x63, 0x52, 0x4c, 0x44,
0x69, 0x73, 0x74, 0x72, 0x69, 0x62, 0x75, 0x74,
0x69, 0x6f, 0x6e, 0x50, 0x6f, 0x69, 0x6e, 0x74
};
static CK_BYTE et3 [] = {
    0xee, 0x8c, 0xd9, 0x40, 0x23, 0xdd, 0x9a, 0xf8,
    0x17, 0xdd, 0xe8, 0x1f, 0x02, 0x25, 0x5b, 0xb3,
    0xaa, 0x6b, 0xd3, 0x21, 0x09, 0x41, 0x95, 0x7e,
    0xea, 0x7e, 0x0e, 0x3c, 0x35, 0x9f, 0x04, 0x5a
};
// CryptoPro gostR3411 A Param Set
static CK_BYTE oid_default [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
};
static CK_BYTE data6 [] = {
    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
};
static CK_BYTE et6 [] = {
    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
};

memset(value, 0, sizeof(value));

mechanism->pParameter = oid;
mechanism->ulParameterLen = sizeof(oid);
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
    data1, strlen(data1), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et1)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
if (memcmp(value, et1, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}

fprintf(stderr, "One step digest...\n");

memset(value, 0, sizeof(value));

mechanism->pParameter = oid_default;
mechanism->ulParameterLen = sizeof(oid_default);
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
```

```
fprintf(stderr ,
    "%4d: C_DigestInit failed, rc = 0x%x\n",
    __LINE__, rc);
return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession ,
    data6, sizeof(data6), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et6)) {
    fprintf(stderr ,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
if (memcmp(value, et6, len) != 0) {
    fprintf(stderr , "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr , "OK\n");

memset(value, 0, sizeof(value));

fprintf(stderr , "Yet another one step digest...\n");
mechanism->pParameter = oid_default;
mechanism->ulParameterLen = sizeof(oid_default);
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession ,
    data2, sizeof(data2), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr ,
```



```
    "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et2)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
if (memcmp(value, et2, len) != 0) {
    fprintf(stderr,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

memset(value, 0, sizeof(value));

fprintf(stderr,
    "The same digest with NULL mechanism parameter...\n");
mechanism->pParameter = NULL_PTR;
mechanism->ulParameterLen = 0;
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(
    hSession, data2, sizeof(data2), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

if (len != sizeof(et2)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n",
        __LINE__, len);
```

```
    return -1;
}
if (memcmp(value, et2, len) != 0) {
    fprintf(stderr,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

memset(value, 0, sizeof(value));

fprintf(stderr, "Get operation state...\n");
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

rc = funcs->C_DigestUpdate(hSession, data3, 39);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

// Save intermediate cryptographic state.
state_len = 0;
rc = funcs->C_GetOperationState(hSession,
    NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetOperationState failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(hSession,
    state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetOperationState failed, rc = 0x%x\n",
```

```
    __LINE__, rc);
    return rc;
}
rc = funcs->C_DigestUpdate(hSession,
    data3 + 39, sizeof(data3) - 39);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_DigestFinal(hSession, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestFinal failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

if (len != sizeof(et3)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
if (memcmp(value, et3, len) != 0) {
    fprintf(stderr,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "Set operation state...\n");
// Restore saved cryptographic state.
rc = funcs->C_SetOperationState(hSession, state, state_len,
    CK_INVALID_HANDLE, CK_INVALID_HANDLE);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_SetOperationState failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
```

```

rc = funcs->C_DigestUpdate(hSession ,
    data3 + 39, sizeof(data3) - 39);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n" ,
        __LINE__ , rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_DigestFinal(hSession , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DigestFinal failed, rc = 0x%x\n" ,
        __LINE__ , rc);
    return rc;
}

if (len != sizeof(et3)) {
    fprintf(stderr ,
        "%4d: Invalid result length: %d\n" , __LINE__ , len);
    return -1;
}
if (memcmp(value , et3 , len) != 0) {
    fprintf(stderr ,
        "%4d: Invalid result value\n" , __LINE__);
    return -2;
}
fprintf(stderr , "OK\n");

free(state);
printf("SUCCESS\n");

return rc;
}

```

ГОСТ Р34.11-2012, 256 бит

Программа вычисляет хеш сообщения по алгоритму ГОСТ Р 34.11-2012 (256 бит) и сравнивает результат с заранее заданным эталоном.

Листинг 3.7: ckm_gostr3411_12_256.c

```
#include "test_common.h"
```

```

CK_RV test_crypto();
CK_RV test_gost3411_2012_256(CK_SESSION_HANDLE hSession);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
    rc = funcs->C_GetMechanismInfo(SlotId,
        CKM_GOSTR3411_12_256, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n===== Mechanism CKM_GOSTR3411_12_256 "
            "not supported =====\n");
    } else {
        fprintf(stderr,
            "\n===== CKM_GOSTR3411_12_256 "
            "test =====\n");
    }
}

```

```

rc = test_gost3411_2012_256(hSession);
if (rc != CKR_OK) {
    fprintf(stderr,
        "ERROR CKM_GOSTR3411_12_256 failed, rc = 0x%x\n",
        rc);
} else {
    fprintf(stderr, "CKM_GOSTR3411_12_256 test passed.\n");
}
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr, "C_CloseSession success\n");
}

out:
    return rc;
}

CK_RV test_gost3411_2012_256(CK_SESSION_HANDLE hSession)
{
    CK_RV rc;
    CK_BYTE value[256];
    CK_BYTE *state;
    CK_ULONG len, state_len;
    CK_OBJECT_HANDLE paramh = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc = {CKM_GOSTR3411_12_256, NULL,
    0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    static CK_BYTE data1[] =
        "Suppose the original message has length = 50 bytes";
    static CK_BYTE et1[] = {
        0xa3, 0xed, 0x85, 0x32, 0x2e, 0x1a, 0x14, 0x79,
        0xb6, 0x05, 0xa7, 0x52, 0xb1, 0xd4, 0x87, 0xfd,
        0x13, 0x88, 0x63, 0xaa, 0x1e, 0xa6, 0x7a, 0x91,
        0xe1, 0x57, 0xaa, 0x53, 0xfc, 0xe7, 0x96, 0xf3,
    };
    static CK_BYTE data2[] = {
        0xb8, 0xe0, 0x45, 0x82, 0x09, 0x28, 0x55, 0xdc,

```

```
0x54, 0x59, 0xca, 0x6b, 0xf8, 0x42, 0xa9, 0x21,
0xb8, 0xef, 0xa7, 0x96, 0x8b, 0x09, 0xea, 0x0e,
0xd5, 0xc3, 0xdf, 0x8c, 0xaf, 0x8a, 0x5e, 0x44,
};
static CK_BYTE et2 [] = {
    0x42, 0x22, 0x71, 0x8a, 0x1a, 0xa7, 0x67, 0x43,
    0xfd, 0x42, 0x45, 0x01, 0x9c, 0xc2, 0xc8, 0x1e,
    0xb4, 0x55, 0x0d, 0x37, 0x0e, 0x17, 0x22, 0x59,
    0x99, 0xc0, 0xd7, 0x00, 0x8b, 0xd8, 0x9f, 0xd3,
};

static CK_BYTE data3 [] = {
    0x30, 0x82, 0x03, 0x9c, 0xa0, 0x03, 0x02, 0x01,
    0x02, 0x02, 0x10, 0x3a, 0xc3, 0xb8, 0xac, 0xec,
    0xfb, 0xd7, 0xae, 0x28, 0xb5, 0x92, 0x9f, 0xd2,
    0xec, 0x4c, 0xf3, 0x30, 0x08, 0x06, 0x06, 0x2a,
    0x85, 0x03, 0x02, 0x02, 0x03, 0x30, 0x81, 0xec,
    0x31, 0x19, 0x30, 0x17, 0x06, 0x09, 0x2a, 0x86,
    0x48, 0x86, 0xf7, 0x0d, 0x01, 0x09, 0x01, 0x16,
    0x0a, 0x67, 0x64, 0x75, 0x63, 0x40, 0x63, 0x61,
    0x2e, 0x72, 0x75, 0x31, 0x0b, 0x30, 0x09, 0x06,
    0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x52, 0x55,
    0x31, 0x15, 0x30, 0x13, 0x06, 0x03, 0x55, 0x04,
    0x07, 0x0c, 0x0c, 0xd0, 0x9c, 0xd0, 0xbe, 0xd1,
    0x81, 0xd0, 0xba, 0xd0, 0xb2, 0xd0, 0xb0, 0x31,
    0x54, 0x30, 0x52, 0x06, 0x03, 0x55, 0x04, 0x0a,
    0x0c, 0x4b, 0xd0, 0x93, 0xd0, 0xbb, 0xd0, 0xb0,
    0xd0, 0xb2, 0xd0, 0xbd, 0xd1, 0x8b, 0xd0, 0xb9,
    0x20, 0xd0, 0x94, 0xd0, 0xbe, 0xd0, 0xb2, 0xd0,
    0xb5, 0xd1, 0x80, 0xd0, 0xb5, 0xd0, 0xbd, 0xd0,
    0xbd, 0xd1, 0x8b, 0xd0, 0xb9, 0x20, 0xd0, 0xa3,
    0xd0, 0xb4, 0xd0, 0xbe, 0xd1, 0x81, 0xd1, 0x82,
    0xd0, 0xbe, 0xd0, 0xb2, 0xd0, 0xb5, 0xd1, 0x80,
    0xd1, 0x8f, 0xd1, 0x8e, 0xd1, 0x89, 0xd0, 0xb8,
    0xd0, 0xb9, 0x20, 0xd0, 0xa6, 0xd0, 0xb5, 0xd0,
    0xbd, 0xd1, 0x82, 0xd1, 0x80, 0x31, 0x2c, 0x30,
    0x2a, 0x06, 0x03, 0x55, 0x04, 0x0b, 0x0c, 0x23,
    0xd0, 0xa6, 0xd0, 0xb5, 0xd0, 0xbd, 0xd1, 0x82,
    0xd1, 0x80, 0x20, 0xd0, 0xa1, 0xd0, 0xb5, 0xd1,
    0x80, 0xd1, 0x82, 0xd0, 0xb8, 0xd1, 0x84, 0xd0,
    0xb8, 0xd0, 0xba, 0xd0, 0xb0, 0xd1, 0x86, 0xd0,
    0xb8, 0xd0, 0xb8, 0x31, 0x27, 0x30, 0x25, 0x06,
    0x03, 0x55, 0x04, 0x03, 0x0c, 0x1e, 0xd0, 0x93,
    0xd0, 0x94, 0xd0, 0xa3, 0xd0, 0xa6, 0x20, 0xd0,
```

```
0xa1,0xd1,0x82,0xd0,0xb0,0xd0,0xbd,0xd0,
0xb4,0xd0,0xb0,0xd1,0x80,0xd1,0x82,0x20,
0xd0,0xa3,0xd0,0xa6,0x30,0x1e,0x17,0x0d,
0x30,0x34,0x30,0x31,0x30,0x39,0x31,0x32,
0x33,0x33,0x32,0x39,0x5a,0x17,0x0d,0x31,
0x34,0x30,0x31,0x30,0x36,0x31,0x32,0x33,
0x33,0x32,0x39,0x5a,0x30,0x81,0xec,0x31,
0x19,0x30,0x17,0x06,0x09,0x2a,0x86,0x48,
0x86,0xf7,0x0d,0x01,0x09,0x01,0x16,0x0a,
0x67,0x64,0x75,0x63,0x40,0x63,0x61,0x2e,
0x72,0x75,0x31,0x0b,0x30,0x09,0x06,0x03,
0x55,0x04,0x06,0x13,0x02,0x52,0x55,0x31,
0x15,0x30,0x13,0x06,0x03,0x55,0x04,0x07,
0x0c,0x0c,0xd0,0x9c,0xd0,0xbe,0xd1,0x81,
0xd0,0xba,0xd0,0xb2,0xd0,0xb0,0x31,0x54,
0x30,0x52,0x06,0x03,0x55,0x04,0x0a,0x0c,
0x4b,0xd0,0x93,0xd0,0xbb,0xd0,0xb0,0xd0,
0xb2,0xd0,0xbd,0xd1,0x8b,0xd0,0xb9,0x20,
0xd0,0x94,0xd0,0xbe,0xd0,0xb2,0xd0,0xb5,
0xd1,0x80,0xd0,0xb5,0xd0,0xbd,0xd0,0xbd,
0xd1,0x8b,0xd0,0xb9,0x20,0xd0,0xa3,0xd0,
0xb4,0xd0,0xbe,0xd1,0x81,0xd1,0x82,0xd0,
0xbe,0xd0,0xb2,0xd0,0xb5,0xd1,0x80,0xd1,
0x8f,0xd1,0x8e,0xd1,0x89,0xd0,0xb8,0xd0,
0xb9,0x20,0xd0,0xa6,0xd0,0xb5,0xd0,0xbd,
0xd1,0x82,0xd1,0x80,0x31,0x2c,0x30,0x2a,
0x06,0x03,0x55,0x04,0x0b,0x0c,0x23,0xd0,
0xa6,0xd0,0xb5,0xd0,0xbd,0xd1,0x82,0xd1,
0x80,0x20,0xd0,0xa1,0xd0,0xb5,0xd1,0x80,
0xd1,0x82,0xd0,0xb8,0xd1,0x84,0xd0,0xb8,
0xd0,0xba,0xd0,0xb0,0xd1,0x86,0xd0,0xb8,
0xd0,0xb8,0x31,0x27,0x30,0x25,0x06,0x03,
0x55,0x04,0x03,0x0c,0x1e,0xd0,0x93,0xd0,
0x94,0xd0,0xa3,0xd0,0xa6,0x20,0xd0,0xa1,
0xd1,0x82,0xd0,0xb0,0xd0,0xbd,0xd0,0xb4,
0xd0,0xb0,0xd1,0x80,0xd1,0x82,0x20,0xd0,
0xa3,0xd0,0xa6,0x30,0x63,0x30,0x1c,0x06,
0x06,0x2a,0x85,0x03,0x02,0x02,0x13,0x30,
0x12,0x06,0x07,0x2a,0x85,0x03,0x02,0x02,
0x23,0x01,0x06,0x07,0x2a,0x85,0x03,0x02,
0x02,0x1e,0x01,0x03,0x43,0x00,0x04,0x40,
0x50,0xab,0x7f,0xc4,0xcc,0x3d,0xd0,0xe2,
0xdd,0x86,0xda,0x19,0x6b,0x14,0x8c,0x78,
0xd9,0xca,0x58,0x67,0x62,0xf3,0xb7,0xba,
```



```

0x7b, 0x2a, 0xda, 0xc1, 0x9c, 0x3f, 0x87, 0xeb,
0xf1, 0xdc, 0xaf, 0x35, 0xad, 0x2d, 0xe1, 0xca,
0xed, 0xc1, 0x8b, 0x82, 0xde, 0xa0, 0x8b, 0x95,
0xdd, 0xa2, 0xac, 0x46, 0x6a, 0x8e, 0xce, 0x5d,
0x5a, 0x16, 0xba, 0x03, 0x29, 0x72, 0x38, 0x27,
0xa3, 0x82, 0x01, 0x14, 0x30, 0x82, 0x01, 0x10,
0x30, 0x5a, 0x06, 0x03, 0x55, 0x1d, 0x11, 0x01,
0x01, 0xff, 0x04, 0x50, 0x30, 0x4e, 0x81, 0x0b,
0x67, 0x64, 0x75, 0x63, 0x63, 0x40, 0x75, 0x63,
0x2e, 0x72, 0x75, 0xa4, 0x3f, 0x30, 0x3d, 0x31,
0x3b, 0x30, 0x39, 0x06, 0x03, 0x55, 0x04, 0x03,
0x0c, 0x32, 0xd0, 0x9f, 0xd0, 0xb5, 0xd1, 0x80,
0xd0, 0xb2, 0xd0, 0xbe, 0xd0, 0xb5, 0x20, 0xd0,
0xa3, 0xd0, 0xbf, 0xd0, 0xbe, 0xd0, 0xbb, 0xd0,
0xbd, 0xd0, 0xbe, 0xd0, 0xbc, 0xd0, 0xbe, 0xd1,
0x87, 0xd0, 0xb5, 0xd0, 0xbd, 0xd0, 0xbd, 0xd0,
0xbe, 0xd0, 0xb5, 0x20, 0xd0, 0x9b, 0xd0, 0xb8,
0xd1, 0x86, 0xd0, 0xbe, 0x30, 0x0f, 0x06, 0x03,
0x55, 0x1d, 0x0f, 0x01, 0x01, 0xff, 0x04, 0x05,
0x03, 0x03, 0x07, 0xc6, 0x00, 0x30, 0x0f, 0x06,
0x03, 0x55, 0x1d, 0x13, 0x01, 0x01, 0xff, 0x04,
0x05, 0x30, 0x03, 0x01, 0x01, 0xff, 0x30, 0x1d,
0x06, 0x03, 0x55, 0x1d, 0x0e, 0x04, 0x16, 0x04,
0x14, 0xb1, 0x6e, 0x0e, 0xa4, 0x40, 0xbc, 0xf0,
0xd9, 0xb6, 0xf7, 0xef, 0xfa, 0xf0, 0x3d, 0xa1,
0x0c, 0xd2, 0x8f, 0xf1, 0xb6, 0x30, 0x71, 0x06,
0x03, 0x55, 0x1d, 0x1f, 0x04, 0x6a, 0x30, 0x68,
0x30, 0x66, 0xa0, 0x64, 0xa0, 0x62, 0x86, 0x60,
0x6c, 0x64, 0x61, 0x70, 0x3a, 0x2f, 0x2f, 0x31,
0x39, 0x32, 0x2e, 0x31, 0x36, 0x38, 0x2e, 0x36,
0x38, 0x2e, 0x37, 0x30, 0x2f, 0x6f, 0x3d, 0x72,
0x6f, 0x6f, 0x74, 0x2c, 0x63, 0x3d, 0x72, 0x75,
0x3f, 0x63, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69,
0x63, 0x61, 0x74, 0x65, 0x52, 0x65, 0x76, 0x6f,
0x63, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x4c, 0x69,
0x73, 0x74, 0x3f, 0x62, 0x61, 0x73, 0x65, 0x3f,
0x6f, 0x62, 0x6a, 0x65, 0x63, 0x74, 0x63, 0x6c,
0x61, 0x73, 0x73, 0x3d, 0x63, 0x52, 0x4c, 0x44,
0x69, 0x73, 0x74, 0x72, 0x69, 0x62, 0x75, 0x74,
0x69, 0x6f, 0x6e, 0x50, 0x6f, 0x69, 0x6e, 0x74
};
static CK_BYTE et3 [] = {
    0xc7, 0x64, 0xc9, 0x1a, 0xc5, 0xcd, 0x56, 0x84,
    0x47, 0xab, 0x2f, 0x9a, 0x6b, 0x9e, 0xc8, 0x69,

```

```

0x18, 0x7f, 0x13, 0x72, 0x8f, 0x4c, 0x8e, 0xb0,
0x30, 0xc8, 0x91, 0xfd, 0x0d, 0x10, 0x73, 0xb0,
};
static CK_BYTE data6 [] = {
    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
};
static CK_BYTE et6 [] = {
    0xe0, 0x05, 0x24, 0xb6, 0x9d, 0xb2, 0x79, 0xbc,
    0x63, 0xf0, 0xd9, 0x0d, 0x40, 0xe1, 0x82, 0x3d,
    0xd1, 0x9f, 0x7a, 0xd6, 0x49, 0x8e, 0x72, 0x45,
    0xab, 0x21, 0x74, 0x03, 0x70, 0x3a, 0x38, 0x2e,
};
// 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 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,
};
// Digest of NULL message, 256 bits, little-endian.
unsigned char et0_32 [] = {
    0x3f, 0x53, 0x9a, 0x21, 0x3e, 0x97, 0xc8, 0x02,
    0xcc, 0x22, 0x9d, 0x47, 0x4c, 0x6a, 0xa3, 0x2a,
    0x82, 0x5a, 0x36, 0x0b, 0x2a, 0x93, 0x3a, 0x94,
    0x9f, 0xd9, 0x25, 0x20, 0x8d, 0x9c, 0xe1, 0xbb
};
static CK_BYTE keyval_33 [] = {
    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,
    0x0A
};
static CK_BYTE keyval_33_hash [] = {
    0x72, 0xc3, 0x40, 0xc5, 0x8f, 0xbb, 0x9d, 0x91,
    0xdf, 0x95, 0x7c, 0xff, 0x0a, 0x87, 0xaa, 0x45,
    0xde, 0xd3, 0xb9, 0x78, 0x12, 0xe8, 0x41, 0xc9,
    0xd4, 0x97, 0x7f, 0xd0, 0xe2, 0xd8, 0xea, 0xb4,
};

fprintf(stderr, "GOST R 34.11-2012 NULL example...\n");
memset(value, 0, sizeof(value));
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_DigestUpdate(hSession,
    NULL, 0);
if (rc != CKR_OK) {

```

```
fprintf(stderr ,
    "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
return rc;
}

rc = funcs->C_DigestFinal(hSession ,
    value , &len);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et0_32)) {
    fprintf(stderr ,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value , len);
if (memcmp(value , et0_32 , len) != 0) {
    fprintf(stderr , "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr , "OK\n");

fprintf(stderr , "GOST R 34.11-2012 M1 example...\n");
memset(value , 0 , sizeof(value));
rc = funcs->C_DigestInit(hSession , mechanism);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession ,
    M1 , sizeof(M1) , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
}
```

```
if (len != sizeof(et1_32)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et1_32, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "GOST R 34.11-2012 M2 example...\n");
memset(value, 0, sizeof(value));
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
    M2, sizeof(M2), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et2_32)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et2_32, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

memset(value, 0, sizeof(value));
```

```
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
    data1, strlen(data1), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et1)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et1, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "One step digest...\n");

memset(value, 0, sizeof(value));

rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
```

```
data6, sizeof(data6), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
if (len != sizeof(et6)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et6, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

memset(value, 0, sizeof(value));

fprintf(stderr, "Yet another one step digest...\n");
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
    data2, sizeof(data2), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et2)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
```

```
if (memcmp(value, et2, len) != 0) {
    fprintf(stderr,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

memset(value, 0, sizeof(value));

fprintf(stderr, "Get operation state...\n");
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

rc = funcs->C_DigestUpdate(hSession, data3, 39);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

// Save intermediate cryptographic state.
state_len = 0;
rc = funcs->C_GetOperationState(hSession,
    NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetOperationState failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(hSession,
    state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetOperationState failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
```



```
}
rc = funcs->C_DigestUpdate(hSession,
    data3 + 39, sizeof(data3) - 39);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_DigestFinal(hSession, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestFinal failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

if (len != sizeof(et3)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et3, len) != 0) {
    fprintf(stderr,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "Set operation state...\n");
// Restore saved cryptographic state.
rc = funcs->C_SetOperationState(hSession, state, state_len,
    CK_INVALID_HANDLE, CK_INVALID_HANDLE);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_SetOperationState failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

rc = funcs->C_DigestUpdate(hSession,
```

```
data3 + 39, sizeof(data3) - 39);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_DigestFinal(hSession, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestFinal failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

if (len != sizeof(et3)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et3, len) != 0) {
    fprintf(stderr,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "keyval_33 example...\n");
memset(value, 0, sizeof(value));
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
    keyval_33, sizeof(keyval_33), value, &len);
if (rc != CKR_OK) {
```

```
fprintf(stderr ,
    "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
return rc;
}

if (len != sizeof(keyval_33_hash)) {
    fprintf(stderr ,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value , len);
if (memcmp(value , keyval_33_hash , len) != 0) {
    fprintf(stderr , "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr , "OK\n");

free(state);
printf("SUCCESS\n");

return rc;
}
```

ГОСТ Р34.11-2012, 512 бит

Программа вычисляет хеш сообщения по алгоритму ГОСТ Р 34.11-2012 (512 бит) и сравнивает результат с заранее заданным эталоном.

Листинг 3.8: ckm_gostr3411_12_512.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost3411_2012_512(CK_SESSION_HANDLE hSession);

int main(int argc , char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc , argv);
    if (rc != CKR_OK) {
        fprintf(stderr , "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
```

```

    fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
    return rc;
}
return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
    rc = funcs->C_GetMechanismInfo(SlotId, CKM_GOSTR3411_12_512, &
        minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n===== Mechanism CKM_GOSTR3411_12_512 not supported
            =====\n");
    } else {
        fprintf(stderr,
            "\n===== CKM_GOSTR3411_12_512 test
            =====\n");
        rc = test_gost3411_2012_512(hSession);
        if (rc != CKR_OK) {
            fprintf(stderr,
                "ERROR CKM_GOSTR3411_12_512 failed, rc = 0x%x\n", rc);
        } else {
            fprintf(stderr, "CKM_GOSTR3411_12_512 test passed.\n");
        }
    }
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}

```

```

    }
    else {
        fprintf(stderr, "C_CloseSession success\n");
    }
}

out:
    return rc;
}

CK_RV test_gost3411_2012_512(CK_SESSION_HANDLE hSession)
{
    CK_RV rc;
    CK_BYTE value[256];
    CK_BYTE *state;
    CK_ULONG len, state_len;
    CK_OBJECT_HANDLE paramh = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc = {CKM_GOSTR3411_12_512, NULL,
    0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    static CK_BYTE data1[] =
        "Suppose the original message has length = 50 bytes";
    static CK_BYTE et1[] = {
        0x27, 0x55, 0x57, 0xa4, 0x7d, 0xcf, 0xcb, 0x82,
        0x35, 0xff, 0x02, 0x9b, 0x74, 0x83, 0x7f, 0x04,
        0x41, 0xef, 0xe4, 0x1a, 0xed, 0x12, 0xc2, 0x07,
        0x31, 0x3e, 0x83, 0xb2, 0x7a, 0xbd, 0x1e, 0x6a,
        0x9d, 0x89, 0x27, 0x13, 0xbc, 0x30, 0xa1, 0x6b,
        0xf9, 0x47, 0xd4, 0x6a, 0x59, 0xbb, 0xbb, 0x3a,
        0x33, 0xee, 0x33, 0x38, 0x53, 0x91, 0xc7, 0x36,
        0x75, 0xe7, 0xd0, 0xc3, 0x60, 0x21, 0x35, 0x40,
    };
    static CK_BYTE data2[] = {
        0xc3, 0x73, 0x0c, 0x5c, 0xbc, 0xca, 0xcf, 0x91,
        0x5a, 0xc2, 0x92, 0x67, 0x6f, 0x21, 0xe8, 0xbd,
        0x4e, 0xf7, 0x53, 0x31, 0xd9, 0x40, 0x5e, 0x5f,
        0x1a, 0x61, 0xdc, 0x31, 0x30, 0xa6, 0x50, 0x11
    };
    static CK_BYTE et2[] = {
        0xf4, 0xae, 0xe0, 0x46, 0x77, 0xea, 0x0e, 0xd4,
        0x8e, 0x92, 0x9d, 0x36, 0x85, 0x0a, 0xc7, 0xba,
        0xe5, 0xef, 0x37, 0x56, 0xaf, 0xcc, 0x25, 0x03,
        0x53, 0xac, 0xf9, 0xf9, 0xf6, 0xf0, 0x0f, 0xf9,
        0x9c, 0xc3, 0xeb, 0xd9, 0x16, 0x3e, 0x40, 0x55,
    };
}

```

```
0x99, 0xa3, 0x02, 0x41, 0x8a, 0x39, 0xb7, 0xf6,  
0xbc, 0xe4, 0x13, 0x75, 0x31, 0xf8, 0xa0, 0xb8,  
0xc2, 0x79, 0x15, 0xaf, 0x16, 0x3c, 0x8d, 0x00,  
};
```

```
static CK_BYTE data3 [] = {  
0x30, 0x82, 0x03, 0x9c, 0xa0, 0x03, 0x02, 0x01,  
0x02, 0x02, 0x10, 0x3a, 0xc3, 0xb8, 0xac, 0xec,  
0xfb, 0xd7, 0xae, 0x28, 0xb5, 0x92, 0x9f, 0xd2,  
0xec, 0x4c, 0xf3, 0x30, 0x08, 0x06, 0x06, 0x2a,  
0x85, 0x03, 0x02, 0x02, 0x03, 0x30, 0x81, 0xec,  
0x31, 0x19, 0x30, 0x17, 0x06, 0x09, 0x2a, 0x86,  
0x48, 0x86, 0xf7, 0x0d, 0x01, 0x09, 0x01, 0x16,  
0x0a, 0x67, 0x64, 0x75, 0x63, 0x40, 0x63, 0x61,  
0x2e, 0x72, 0x75, 0x31, 0x0b, 0x30, 0x09, 0x06,  
0x03, 0x55, 0x04, 0x06, 0x13, 0x02, 0x52, 0x55,  
0x31, 0x15, 0x30, 0x13, 0x06, 0x03, 0x55, 0x04,  
0x07, 0x0c, 0x0c, 0xd0, 0x9c, 0xd0, 0xbe, 0xd1,  
0x81, 0xd0, 0xba, 0xd0, 0xb2, 0xd0, 0xb0, 0x31,  
0x54, 0x30, 0x52, 0x06, 0x03, 0x55, 0x04, 0x0a,  
0x0c, 0x4b, 0xd0, 0x93, 0xd0, 0xbb, 0xd0, 0xb0,  
0xd0, 0xb2, 0xd0, 0xbd, 0xd1, 0x8b, 0xd0, 0xb9,  
0x20, 0xd0, 0x94, 0xd0, 0xbe, 0xd0, 0xb2, 0xd0,  
0xb5, 0xd1, 0x80, 0xd0, 0xb5, 0xd0, 0xbd, 0xd0,  
0xbd, 0xd1, 0x8b, 0xd0, 0xb9, 0x20, 0xd0, 0xa3,  
0xd0, 0xb4, 0xd0, 0xbe, 0xd1, 0x81, 0xd1, 0x82,  
0xd0, 0xbe, 0xd0, 0xb2, 0xd0, 0xb5, 0xd1, 0x80,  
0xd1, 0x8f, 0xd1, 0x8e, 0xd1, 0x89, 0xd0, 0xb8,  
0xd0, 0xb9, 0x20, 0xd0, 0xa6, 0xd0, 0xb5, 0xd0,  
0xbd, 0xd1, 0x82, 0xd1, 0x80, 0x31, 0x2c, 0x30,  
0x2a, 0x06, 0x03, 0x55, 0x04, 0x0b, 0x0c, 0x23,  
0xd0, 0xa6, 0xd0, 0xb5, 0xd0, 0xbd, 0xd1, 0x82,  
0xd1, 0x80, 0x20, 0xd0, 0xa1, 0xd0, 0xb5, 0xd1,  
0x80, 0xd1, 0x82, 0xd0, 0xb8, 0xd1, 0x84, 0xd0,  
0xb8, 0xd0, 0xba, 0xd0, 0xb0, 0xd1, 0x86, 0xd0,  
0xb8, 0xd0, 0xb8, 0x31, 0x27, 0x30, 0x25, 0x06,  
0x03, 0x55, 0x04, 0x03, 0x0c, 0x1e, 0xd0, 0x93,  
0xd0, 0x94, 0xd0, 0xa3, 0xd0, 0xa6, 0x20, 0xd0,  
0xa1, 0xd1, 0x82, 0xd0, 0xb0, 0xd0, 0xbd, 0xd0,  
0xb4, 0xd0, 0xb0, 0xd1, 0x80, 0xd1, 0x82, 0x20,  
0xd0, 0xa3, 0xd0, 0xa6, 0x30, 0x1e, 0x17, 0xd0,  
0x30, 0x34, 0x30, 0x31, 0x30, 0x39, 0x31, 0x32,  
0x33, 0x33, 0x32, 0x39, 0x5a, 0x17, 0xd0, 0x31,  
0x34, 0x30, 0x31, 0x30, 0x36, 0x31, 0x32, 0x33,
```

```
0x33,0x32,0x39,0x5a,0x30,0x81,0xec,0x31,
0x19,0x30,0x17,0x06,0x09,0x2a,0x86,0x48,
0x86,0xf7,0x0d,0x01,0x09,0x01,0x16,0x0a,
0x67,0x64,0x75,0x63,0x40,0x63,0x61,0x2e,
0x72,0x75,0x31,0x0b,0x30,0x09,0x06,0x03,
0x55,0x04,0x06,0x13,0x02,0x52,0x55,0x31,
0x15,0x30,0x13,0x06,0x03,0x55,0x04,0x07,
0x0c,0x0c,0xd0,0x9c,0xd0,0xbe,0xd1,0x81,
0xd0,0xba,0xd0,0xb2,0xd0,0xb0,0x31,0x54,
0x30,0x52,0x06,0x03,0x55,0x04,0x0a,0x0c,
0x4b,0xd0,0x93,0xd0,0xbb,0xd0,0xb0,0xd0,
0xb2,0xd0,0xbd,0xd1,0x8b,0xd0,0xb9,0x20,
0xd0,0x94,0xd0,0xbe,0xd0,0xb2,0xd0,0xb5,
0xd1,0x80,0xd0,0xb5,0xd0,0xbd,0xd0,0xbd,
0xd1,0x8b,0xd0,0xb9,0x20,0xd0,0xa3,0xd0,
0xb4,0xd0,0xbe,0xd1,0x81,0xd1,0x82,0xd0,
0xbe,0xd0,0xb2,0xd0,0xb5,0xd1,0x80,0xd1,
0x8f,0xd1,0x8e,0xd1,0x89,0xd0,0xb8,0xd0,
0xb9,0x20,0xd0,0xa6,0xd0,0xb5,0xd0,0xbd,
0xd1,0x82,0xd1,0x80,0x31,0x2c,0x30,0x2a,
0x06,0x03,0x55,0x04,0x0b,0x0c,0x23,0xd0,
0xa6,0xd0,0xb5,0xd0,0xbd,0xd1,0x82,0xd1,
0x80,0x20,0xd0,0xa1,0xd0,0xb5,0xd1,0x80,
0xd1,0x82,0xd0,0xb8,0xd1,0x84,0xd0,0xb8,
0xd0,0xba,0xd0,0xb0,0xd1,0x86,0xd0,0xb8,
0xd0,0xb8,0x31,0x27,0x30,0x25,0x06,0x03,
0x55,0x04,0x03,0x0c,0x1e,0xd0,0x93,0xd0,
0x94,0xd0,0xa3,0xd0,0xa6,0x20,0xd0,0xa1,
0xd1,0x82,0xd0,0xb0,0xd0,0xbd,0xd0,0xb4,
0xd0,0xb0,0xd1,0x80,0xd1,0x82,0x20,0xd0,
0xa3,0xd0,0xa6,0x30,0x63,0x30,0x1c,0x06,
0x06,0x2a,0x85,0x03,0x02,0x02,0x13,0x30,
0x12,0x06,0x07,0x2a,0x85,0x03,0x02,0x02,
0x23,0x01,0x06,0x07,0x2a,0x85,0x03,0x02,
0x02,0x1e,0x01,0x03,0x43,0x00,0x04,0x40,
0x50,0xab,0x7f,0xc4,0xcc,0x3d,0xd0,0xe2,
0xdd,0x86,0xda,0x19,0x6b,0x14,0x8c,0x78,
0xd9,0xca,0x58,0x67,0x62,0xf3,0xb7,0xba,
0x7b,0x2a,0xda,0xc1,0x9c,0x3f,0x87,0xeb,
0xf1,0xdc,0xaf,0x35,0xad,0x2d,0xe1,0xca,
0xed,0xc1,0x8b,0x82,0xde,0xa0,0x8b,0x95,
0xdd,0xa2,0xac,0x46,0x6a,0x8e,0xce,0x5d,
0x5a,0x16,0xba,0x03,0x29,0x72,0x38,0x27,
0xa3,0x82,0x01,0x14,0x30,0x82,0x01,0x10,
```

```

0x30, 0x5a, 0x06, 0x03, 0x55, 0x1d, 0x11, 0x01,
0x01, 0xff, 0x04, 0x50, 0x30, 0x4e, 0x81, 0x0b,
0x67, 0x64, 0x75, 0x63, 0x63, 0x40, 0x75, 0x63,
0x2e, 0x72, 0x75, 0xa4, 0x3f, 0x30, 0x3d, 0x31,
0x3b, 0x30, 0x39, 0x06, 0x03, 0x55, 0x04, 0x03,
0x0c, 0x32, 0xd0, 0x9f, 0xd0, 0xb5, 0xd1, 0x80,
0xd0, 0xb2, 0xd0, 0xbe, 0xd0, 0xb5, 0x20, 0xd0,
0xa3, 0xd0, 0xbf, 0xd0, 0xbe, 0xd0, 0xbb, 0xd0,
0xbd, 0xd0, 0xbe, 0xd0, 0xbc, 0xd0, 0xbe, 0xd1,
0x87, 0xd0, 0xb5, 0xd0, 0xbd, 0xd0, 0xbd, 0xd0,
0xbe, 0xd0, 0xb5, 0x20, 0xd0, 0x9b, 0xd0, 0xb8,
0xd1, 0x86, 0xd0, 0xbe, 0x30, 0x0f, 0x06, 0x03,
0x55, 0x1d, 0x0f, 0x01, 0x01, 0xff, 0x04, 0x05,
0x03, 0x03, 0x07, 0xc6, 0x00, 0x30, 0x0f, 0x06,
0x03, 0x55, 0x1d, 0x13, 0x01, 0x01, 0xff, 0x04,
0x05, 0x30, 0x03, 0x01, 0x01, 0xff, 0x30, 0x1d,
0x06, 0x03, 0x55, 0x1d, 0x0e, 0x04, 0x16, 0x04,
0x14, 0xb1, 0x6e, 0x0e, 0xa4, 0x40, 0xbc, 0xf0,
0xd9, 0xb6, 0xf7, 0xef, 0xfa, 0xf0, 0x3d, 0xa1,
0x0c, 0xd2, 0x8f, 0xf1, 0xb6, 0x30, 0x71, 0x06,
0x03, 0x55, 0x1d, 0x1f, 0x04, 0x6a, 0x30, 0x68,
0x30, 0x66, 0xa0, 0x64, 0xa0, 0x62, 0x86, 0x60,
0x6c, 0x64, 0x61, 0x70, 0x3a, 0x2f, 0x2f, 0x31,
0x39, 0x32, 0x2e, 0x31, 0x36, 0x38, 0x2e, 0x36,
0x38, 0x2e, 0x37, 0x30, 0x2f, 0x6f, 0x3d, 0x72,
0x6f, 0x6f, 0x74, 0x2c, 0x63, 0x3d, 0x72, 0x75,
0x3f, 0x63, 0x65, 0x72, 0x74, 0x69, 0x66, 0x69,
0x63, 0x61, 0x74, 0x65, 0x52, 0x65, 0x76, 0x6f,
0x63, 0x61, 0x74, 0x69, 0x6f, 0x6e, 0x4c, 0x69,
0x73, 0x74, 0x3f, 0x62, 0x61, 0x73, 0x65, 0x3f,
0x6f, 0x62, 0x6a, 0x65, 0x63, 0x74, 0x63, 0x6c,
0x61, 0x73, 0x73, 0x3d, 0x63, 0x52, 0x4c, 0x44,
0x69, 0x73, 0x74, 0x72, 0x69, 0x62, 0x75, 0x74,
0x69, 0x6f, 0x6e, 0x50, 0x6f, 0x69, 0x6e, 0x74
};
static CK_BYTE et3 [] = {
    0x83, 0x84, 0xb1, 0xed, 0x9a, 0xd7, 0x04, 0x34,
    0x8c, 0xa9, 0x3f, 0xd1, 0xdb, 0x6b, 0x4b, 0xb1,
    0x96, 0x4b, 0xce, 0x40, 0x9c, 0x95, 0x85, 0x47,
    0xb0, 0x16, 0x2a, 0x12, 0x2c, 0xfa, 0xd5, 0xd2,
    0x9d, 0x9f, 0xdb, 0x43, 0x10, 0x6e, 0x6a, 0xef,
    0x39, 0xf2, 0xa9, 0x1e, 0x1a, 0x6a, 0xd7, 0xd4,
    0xed, 0x94, 0xf6, 0x84, 0xc9, 0x41, 0x6e, 0x92,
    0x83, 0xde, 0x14, 0xbf, 0xfa, 0x4c, 0x21, 0xb8,

```



```
};
static CK_BYTE data6 [] = {
    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
};
static CK_BYTE et6 [] = {
    0x89, 0x8c, 0xd4, 0xf2, 0x23, 0x32, 0xf6, 0x2f,
    0x2b, 0xae, 0x32, 0xaf, 0x46, 0xc1, 0x1d, 0x4e,
    0x64, 0x13, 0xff, 0x89, 0xaa, 0xd4, 0xeb, 0x53,
    0xb4, 0xa2, 0xd8, 0x85, 0x4b, 0x0e, 0xf0, 0xe1,
    0xd2, 0xbe, 0x75, 0x01, 0xec, 0x40, 0x2e, 0x98,
    0x2a, 0xb2, 0x95, 0xb8, 0xec, 0x25, 0x72, 0xe8,
    0xa8, 0x0d, 0x1a, 0xe3, 0xbf, 0xff, 0x17, 0x85,
    0xad, 0xcc, 0x42, 0x8f, 0x4f, 0xbb, 0x76, 0x0b,
};
// 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_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_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 ,
};
// Digest of NULL message , 512 bits , little-endian.
unsigned char et0_64 [] = {
    0x8e , 0x94 , 0x5d , 0xa2 , 0x09 , 0xaa , 0x86 , 0x9f ,
    0x04 , 0x55 , 0x92 , 0x85 , 0x29 , 0xbc , 0xae , 0x46 ,
    0x79 , 0xe9 , 0x87 , 0x3a , 0xb7 , 0x07 , 0xb5 , 0x53 ,
    0x15 , 0xf5 , 0x6c , 0xeb , 0x98 , 0xbe , 0xf0 , 0xa7 ,
    0x36 , 0x2f , 0x71 , 0x55 , 0x28 , 0x35 , 0x6e , 0xe8 ,
    0x3c , 0xda , 0x5f , 0x2a , 0xac , 0x4c , 0x6a , 0xd2 ,
    0xba , 0x3a , 0x71 , 0x5c , 0x1b , 0xcd , 0x81 , 0xcb ,
    0x8e , 0x9f , 0x90 , 0xbf , 0x4c , 0x1c , 0x1a , 0x8a
};

fprintf(stderr , "GOST R 34.11-2012 NULL example...\n");
memset(value , 0 , sizeof(value));
rc = funcs->C_DigestInit(hSession , mechanism);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DigestInit failed, rc = 0x%x\n" ,
        __LINE__ , rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_DigestUpdate(hSession ,
    NULL , 0);

```

```
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

rc = funcs->C_DigestFinal(hSession,
    value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et0_64)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et0_64, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "GOST R 34.11-2012 M1 example...\n");
memset(value, 0, sizeof(value));
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
    M1, sizeof(M1), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
```

```
if (len != sizeof(et1_64)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et1_64, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "GOST R 34.11-2012 M2 example...\n");
memset(value, 0, sizeof(value));
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
    M2, sizeof(M2), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et2_64)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et2_64, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");
```

```
memset(value, 0, sizeof(value));

rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession,
    data1, strlen(data1), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et1)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et1, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "One step digest...\n");

memset(value, 0, sizeof(value));

rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
```

```
rc = funcs->C_Digest(hSession ,
    data6, sizeof(data6), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et6)) {
    fprintf(stderr ,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et6, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

memset(value, 0, sizeof(value));

fprintf(stderr, "Yet another one step digest...\n");
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Digest(hSession ,
    data2, sizeof(data2), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_Digest failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

if (len != sizeof(et2)) {
    fprintf(stderr ,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
```

```
}
print_hex(value, len);
if (memcmp(value, et2, len) != 0) {
    fprintf(stderr,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

memset(value, 0, sizeof(value));

fprintf(stderr, "Get operation state...\n");
rc = funcs->C_DigestInit(hSession, mechanism);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestInit failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

rc = funcs->C_DigestUpdate(hSession, data3, 39);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

// Save intermediate cryptographic state.
state_len = 0;
rc = funcs->C_GetOperationState(hSession,
    NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetOperationState failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(hSession,
    state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetOperationState failed, rc = 0x%x\n",
```

```
    __LINE__, rc);
    return rc;
}
rc = funcs->C_DigestUpdate(hSession,
    data3 + 39, sizeof(data3) - 39);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_DigestFinal(hSession, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DigestFinal failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

if (len != sizeof(et3)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value, len);
if (memcmp(value, et3, len) != 0) {
    fprintf(stderr,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr, "OK\n");

fprintf(stderr, "Set operation state...\n");
// Restore saved cryptographic state.
rc = funcs->C_SetOperationState(hSession, state, state_len,
    CK_INVALID_HANDLE, CK_INVALID_HANDLE);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_SetOperationState failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
}
```



```
rc = funcs->C_DigestUpdate(hSession ,
    data3 + 39, sizeof(data3) - 39);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DigestUpdate failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_DigestFinal(hSession , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DigestFinal failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

if (len != sizeof(et3)) {
    fprintf(stderr ,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
print_hex(value , len);
if (memcmp(value , et3 , len) != 0) {
    fprintf(stderr ,
        "%4d: Invalid result value\n", __LINE__);
    return -2;
}
fprintf(stderr , "OK\n");

free(state);
printf("SUCCESS\n");

return rc;
}
```

3.3.5 Генерация HMAC

ГОСТ Р34.11-94

В данном примере демонстрируется генерация и проверка HMAC механизмом СКМ_GOSTR3411_HMAC. Заметим, что длина значения исходного ключа при генерации

НМАС не обязана быть равной 32-м байтам.

Генерация НМАС - это контекстная операция в сессии, поэтому ее промежуточное состояние может быть сохранено и восстановлено функциями `C_GetOperationState`, `C_SetOperationState`.

Заметим, что ключ, используемый в контексте генерации НМАС, рассматривается, как ключ аутентификации, а не как ключ шифрования, и поэтому задается в последнем параметре функции `C_SetOperationState`.

Размер состояния контекста генерации НМАС зависит от реализации и в разных версиях библиотеки может оказаться различным, поэтому не следует надеяться на фиксированный размер буфера при сохранении этого состояния с помощью функции `C_GetOperationState`.

Листинг 3.9: `ckm_gostr3411_hmac.c`

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost3411_hmac(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
```

```

    fprintf(stderr ,
        "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
    goto out;
}
fprintf(stderr , "C_OpenSession success\n");

rc = funcs->C_GetMechanismInfo(SlotId , CKM_GOSTR3411_HMAC, &
    minfo);
if (rc != CKR_OK) {
    fprintf(stderr ,
"\n===== Mechanism CKM_GOSTR3411_HMAC not supported
=====\\n");
} else {
    fprintf(stderr ,
"\n===== CKM_GOSTR3411_HMAC test
=====\\n");
    rc = test_gost3411_hmac(hSession);
    if (rc != CKR_OK) {
        fprintf(stderr ,
            "ERROR CKM_GOSTR3411_HMAC failed, rc = 0x%x\n", rc);
    } else {
        fprintf(stderr , "CKM_GOSTR3411_HMAC test passed.\\n");
    }
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr ,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr , "C_CloseSession success\n");
}

out:
    return rc;
}

CK_RV test_gost3411_hmac(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[256];
    CK_ULONG len;
    CK_MECHANISM mechanism_desc =

```

```

{CKM_GOSTR3411_HMAC, NULL, 0};
CK_MECHANISM_PTR mechanism = &mechanism_desc;

CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
static CK_KEY_TYPE key_type = CKK_GENERIC_SECRET;
static CK_BBOOL ltrue = CK_TRUE;

static CK_BYTE keyval1 [] = {
    0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,0x0b,
    0x0b,0x0b,0x0b,0x0b,
};
static CK_BYTE data1 [] = "Hi There";
static CK_BYTE et1 [] = {
    0x34,0x4f,0x17,0xcd,0xa0,0xfa,0x9a,0x56,
    0xdb,0x24,0xed,0x7c,0xf3,0xaa,0xcd,0xe1,
    0xd1,0x26,0xb9,0xe2,0x4e,0xf3,0x92,0xf2,
    0x31,0x77,0x0e,0x3e,0xa8,0x6d,0xde,0x1d,
};

static CK_BYTE keyval2 [] = { 'J','e','f','e' };
static CK_BYTE data2 [] = "what do ya want for nothing?";
static CK_BYTE et2 [] = {
    0x00,0x61,0x75,0x04,0x8e,0x45,0x72,0xac,
    0x61,0x85,0xfa,0xf5,0xff,0xae,0x49,0x62,
    0x97,0xf9,0x99,0x3f,0xf1,0xab,0xb7,0x05,
    0xce,0x43,0xda,0x09,0x51,0x56,0x8d,0x9a,
};

static CK_BYTE keyval3 [] = {
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
    0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,0xaa,
};
static CK_BYTE data3 [] =
    "Test Using Larger Than Block-Size Key - Hash Key First";

```

```

static CK_BYTE et3 [] = {
    0x91, 0x77, 0xa0, 0x5f, 0xee, 0xca, 0xfa, 0x5c,
    0xbb, 0x14, 0x8b, 0x17, 0x61, 0x63, 0x70, 0xb5,
    0xd5, 0x9b, 0x23, 0x4c, 0x56, 0x7e, 0x26, 0x0d,
    0xda, 0xe2, 0x67, 0x2a, 0x5f, 0xd3, 0x45, 0xfa,
};
// CryptoPro 3411 HASH1 default Param Set
static CK_BYTE oid [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01,
};
static CK_ATTRIBUTE key_template [] = {
    { CKA_VALUE, NULL_PTR, 0 },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_SIGN, &ltrue, sizeof(ltrue) },
    { CKA_VERIFY, &ltrue, sizeof(ltrue) }
};

static CK_BYTE data4 [] = {
    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
};

static CK_BYTE keyval_32 [] = {
    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
};
// For big-endian byte order
static CK_BYTE et_32 [] = {
    0x52, 0x67, 0xfb, 0xa5, 0xbf, 0x8c, 0x73, 0xaf,
    0x26, 0x2c, 0xc2, 0xa0, 0x45, 0x61, 0x86, 0x2e,
    0x28, 0xce, 0xd8, 0xe9, 0x6f, 0x0a, 0x85, 0xf8,
    0xf8, 0x8b, 0x59, 0x40, 0xb8, 0xf5, 0x4a, 0x56
};
// For little-endian byte order
static CK_BYTE et_32_le [] = {
    0x6e, 0x18, 0xbf, 0x2e, 0x66, 0x0c, 0x89, 0xb4,

```

```

    0x85, 0xd7, 0x77, 0x25, 0x90, 0x4b, 0x9c, 0x7a,
    0xbb, 0x85, 0x0e, 0x87, 0x90, 0x09, 0xb5, 0xfa,
    0xee, 0x9a, 0x41, 0x92, 0xac, 0x81, 0xc8, 0x67
};
static CK_BYTE keyval_31 [] = {
    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
};
// For big-endian byte order
static CK_BYTE et_31 [] = {
    0x98, 0x24, 0x2e, 0xd7, 0x01, 0x14, 0xc6, 0x94,
    0xbf, 0xfd, 0x10, 0x7b, 0x2f, 0x7e, 0xc3, 0xc7,
    0x87, 0x14, 0x61, 0xaf, 0x7f, 0x39, 0xf4, 0x05,
    0xe7, 0x2b, 0xfb, 0x63, 0x36, 0x15, 0xf1, 0xce
};
// For little-endian byte order
static CK_BYTE et_31_le [] = {
    0xd7, 0xee, 0x31, 0x90, 0xa7, 0x8e, 0xd2, 0xd3,
    0x25, 0xf7, 0x3c, 0xf5, 0xfc, 0xe4, 0x73, 0x54,
    0x11, 0x34, 0x2f, 0x5c, 0x17, 0x98, 0xc9, 0xea,
    0x25, 0xb2, 0x8e, 0x90, 0xd3, 0x2c, 0x64, 0x16
};
static CK_BYTE keyval_33 [] = {
    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,
    0x0A
};
static CK_BYTE keyval_33_hash [] = {
    0x10, 0xff, 0xed, 0x16, 0x36, 0x4d, 0xa5, 0x3a,
    0x07, 0xc9, 0xba, 0x00, 0x0c, 0xb5, 0x55, 0x31,
    0xab, 0x53, 0xda, 0xf7, 0x1c, 0x1f, 0xfe, 0x31,
    0xad, 0x48, 0x81, 0xaf, 0xa1, 0x31, 0xae, 0xbe
};
// For big-endian byte order
static CK_BYTE et_33 [] = {
    0x18, 0x8f, 0x6c, 0xc3, 0x3f, 0x19, 0xbe, 0x1b,
    0x00, 0xa0, 0x67, 0x80, 0x90, 0xdb, 0xb2, 0x2b,
    0x5a, 0xaa, 0xb3, 0xec, 0x4e, 0x97, 0x5f, 0x6a,
    0xa5, 0xe5, 0x2b, 0xf3, 0x39, 0x57, 0x72, 0xeb
};
};

```

```
// For little-endian byte order
static CK_BYTE et_33_le[] = {
    0xe5, 0x60, 0x6b, 0x35, 0x61, 0x72, 0x53, 0x9e,
    0x69, 0x61, 0x67, 0x3a, 0x85, 0xaa, 0xad, 0xb7,
    0xe2, 0x99, 0x79, 0x06, 0x6d, 0x22, 0x13, 0x47,
    0xa4, 0xe0, 0xa6, 0xbe, 0xec, 0x0f, 0xd7, 0xb9
};
CK_BYTE *state = NULL;
CK_ULONG state_len = 0;

key_template[0].pValue = keyval_32;
key_template[0].ulValueLen = sizeof(keyval_32);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

mechanism->pParameter = oid;
mechanism->ulParameterLen = sizeof(oid);
rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4),
    value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_32_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -2;
    goto end;
}
if (memcmp(value, et_32_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
}
```

```
rc = -3;
goto end;
}
printf("Sign 32_le OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4),
    value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("Verify 32_le OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval_31;
key_template[0].ulValueLen = sizeof(keyval_31);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

// Используется умалчиваемый набор параметров
mechanism->pParameter = NULL;
mechanism->ulParameterLen = 0;
rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}
```



```
memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4),
    value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_31_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -4;
    goto end;
}
if (memcmp(value, et_31_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -5;
    goto end;
}
printf("Sign 31_le OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4),
    value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("Verify 31_le OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval_33_hash;
key_template[0].ulValueLen = sizeof(keyval_33_hash);
```

```
rc = funcs->C_CreateObject(sess ,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE) ,
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

mechanism->pParameter = 0;
mechanism->ulParameterLen = 0;
rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_SignUpdate(sess, data4, 7);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignUpdate failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_GetOperationState(sess, NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
state = (CK_BYTE *)malloc(state_len);
rc = funcs->C_GetOperationState(sess, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SetOperationState(sess, state, state_len,
    CK_INVALID_HANDLE, keyh);
if (rc != CKR_OK) {
    fprintf(stderr,
        "C_SetOperationState failed: 0x%x\n", rc);
    goto end;
}
```

```
}

rc = funcs->C_SignUpdate(sess , data4+7, sizeof(data4)-7);
if (rc != CKR_OK) {
    fprintf(stderr , "C_SignUpdate failed: 0x%x\n" , rc);
    goto end;
}

rc = funcs->C_SignFinal(sess , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr , "C_SignFinal failed: 0x%x\n" , rc);
    goto end;
}

if (len != sizeof(et_33_le)) {
    fprintf(stderr , "Invalid length: %d\n" , len);
    rc = -6;
    goto end;
}
if (memcmp(value , et_33_le , len) != 0) {
    fprintf(stderr , "Invalid value\n");
    rc = -7;
    goto end;
}
printf("Sign 33_le multipart OK\n");

mechanism->pParameter = 0;
mechanism->ulParameterLen = 0;
rc = funcs->C_SignInit(sess , mechanism , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_SignInit failed: 0x%x\n" , rc);
    goto end;
}

rc = funcs->C_SetOperationState(sess , state , state_len ,
    CK_INVALID_HANDLE, keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_SetOperationState failed: 0x%x\n" , rc);
    goto end;
}

rc = funcs->C_SignUpdate(sess , data4+7, sizeof(data4)-7);
if (rc != CKR_OK) {
    fprintf(stderr , "C_SignUpdate failed: 0x%x\n" , rc);
```

```
    goto end;
}

rc = funcs->C_SignFinal(sess, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignFinal failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_33_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -6;
    goto end;
}
if (memcmp(value, et_33_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -7;
    goto end;
}
printf("Restored Sign 33_le multipart OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("Verify 33_le OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval_33;
key_template[0].ulValueLen = sizeof(keyval_33);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
```

```
&keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_33_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -8;
    goto end;
}
if (memcmp(value, et_33_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -9;
    goto end;
}

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_DestroyObject(sess, keyh);
```

```
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval1;
key_template[0].ulValueLen = sizeof(keyval1);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

mechanism->pParameter = 0;
mechanism->ulParameterLen = 0;

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data1, strlen(data1), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et1)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -10;
    goto end;
}
if (memcmp(value, et1, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -11;
    goto end;
}

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
```

```
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data1, strlen(data1), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("SUCCESS\n");
rc = CKR_OK;
end:
if (keyh != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess, keyh);
}

return rc;
}
```

ГОСТ Р34.11-2012, 256 бит

В данном примере демонстрируется генерация и проверка HMAC механизмом СКМ\GOSTR3411\12\256_HMAC.

Листинг 3.10: ckm_gostr3411_12_256_hmac.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost3411_12_256_hmac(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
}
```

```
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession( SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");

    rc = funcs->C_GetMechanismInfo( SlotId,
        CKM_GOSTR3411_12_256_HMAC, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n===== Mechanism CKM_GOSTR3411_12_256_HMAC "
            "not supported =====\n");
    } else {
        fprintf(stderr,
            "\n===== CKM_GOSTR3411_12_256_HMAC "
            "test =====\n");
        rc = test_gost3411_12_256_hmac(hSession);
        if (rc != CKR_OK) {
            fprintf(stderr,
                "ERROR CKM_GOSTR3411_12_256_HMAC failed, rc = 0x%x\n",
                rc);
        } else {
            fprintf(stderr,
                "CKM_GOSTR3411_12_256_HMAC test passed.\n");
        }
    }
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
```



```

    }
    else {
        fprintf(stderr, "C_CloseSession success\n");
    }
}

out:
    return rc;
}

CK_RV test_gost3411_12_256_hmac(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[256];
    CK_ULONG len;
    CK_MECHANISM mechanism_desc =
        {CKM_GOSTR3411_12_256_HMAC, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_KEY_TYPE key_type = CKK_GENERIC_SECRET;
    static CK_BBOOL ltrue = CK_TRUE;

    static CK_BYTE keyval1[] = {
        0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
        0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
        0x0b, 0x0b, 0x0b, 0x0b,
    };

    static CK_BYTE data1[] = "Hi There";
    static CK_BYTE et1[] = {
        0x79, 0x8b, 0xd0, 0x9e, 0xf7, 0xad, 0x67, 0x64,
        0x6a, 0x3b, 0x74, 0xfe, 0x48, 0xd7, 0x31, 0x73,
        0x50, 0xa8, 0x6f, 0x07, 0x7d, 0x05, 0xef, 0x78,
        0xcb, 0xe9, 0xba, 0x3c, 0x9e, 0xf3, 0xa4, 0x11,
    };

    static CK_BYTE keyval2[] = { 'J', 'e', 'f', 'e' };
    static CK_BYTE data2[] = "what do ya want for nothing?";
    static CK_BYTE et2[] = {
        0x00, 0x61, 0x75, 0x04, 0x8e, 0x45, 0x72, 0xac,
        0x61, 0x85, 0xfa, 0xf5, 0xff, 0xae, 0x49, 0x62,
        0x97, 0xf9, 0x99, 0x3f, 0xf1, 0xab, 0xb7, 0x05,
        0xce, 0x43, 0xda, 0x09, 0x51, 0x56, 0x8d, 0x9a,
    };
};

```

```

static CK_BYTE keyval3 [] = {
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
};
static CK_BYTE data3 [] =
    "Test Using Larger Than Block-Size Key - Hash Key First";
static CK_BYTE et3 [] = {
    0x91, 0x77, 0xa0, 0x5f, 0xee, 0xca, 0xfa, 0x5c,
    0xbb, 0x14, 0x8b, 0x17, 0x61, 0x63, 0x70, 0xb5,
    0xd5, 0x9b, 0x23, 0x4c, 0x56, 0x7e, 0x26, 0x0d,
    0xda, 0xe2, 0x67, 0x2a, 0x5f, 0xd3, 0x45, 0xfa,
};
static CK_ATTRIBUTE key_template [] = {
    { CKA_VALUE, NULL_PTR, 0 },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_SIGN, &ltrue, sizeof(ltrue) },
    { CKA_VERIFY, &ltrue, sizeof(ltrue) }
};

static CK_BYTE data4 [] = {
    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
};
static CK_BYTE keyval_32 [] = {
    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
};

```

```

};
// For big-endian byte order
static CK_BYTE et_32 [] = {
    0x52, 0x67, 0xfb, 0xa5, 0xbf, 0x8c, 0x73, 0xaf,
    0x26, 0x2c, 0xc2, 0xa0, 0x45, 0x61, 0x86, 0x2e,
    0x28, 0xce, 0xd8, 0xe9, 0x6f, 0x0a, 0x85, 0xf8,
    0xf8, 0x8b, 0x59, 0x40, 0xb8, 0xf5, 0x4a, 0x56
};
// For little-endian byte order
static CK_BYTE et_32_le [] = {
    0xab, 0x42, 0x71, 0xd2, 0x71, 0xd0, 0xbb, 0x78,
    0x67, 0x5e, 0xe8, 0x10, 0x33, 0x7e, 0x5c, 0xd3,
    0xcf, 0x1c, 0x3c, 0x2b, 0x7d, 0x12, 0xd1, 0xdd,
    0xf1, 0xec, 0x31, 0xf9, 0xfc, 0x00, 0x5b, 0xe2,
};
static CK_BYTE keyval_31 [] = {
    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
};
// For big-endian byte order
static CK_BYTE et_31 [] = {
    0x98, 0x24, 0x2e, 0xd7, 0x01, 0x14, 0xc6, 0x94,
    0xbf, 0xfd, 0x10, 0x7b, 0x2f, 0x7e, 0xc3, 0xc7,
    0x87, 0x14, 0x61, 0xaf, 0x7f, 0x39, 0xf4, 0x05,
    0xe7, 0x2b, 0xfb, 0x63, 0x36, 0x15, 0xf1, 0xce
};
// For little-endian byte order
static CK_BYTE et_31_le [] = {
    0x04, 0x9a, 0xe0, 0xf4, 0xa2, 0xba, 0xac, 0x75,
    0x82, 0xd5, 0x5d, 0x46, 0x45, 0x47, 0x17, 0x71,
    0xec, 0x77, 0xbc, 0x24, 0xa2, 0xb8, 0x5c, 0xbd,
    0xaa, 0x53, 0x0c, 0xb6, 0x85, 0x50, 0x4c, 0x9c,
};
static CK_BYTE keyval_33 [] = {
    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,
    0x0A
};
static CK_BYTE keyval_33_hash [] = {
    0x72, 0xc3, 0x40, 0xc5, 0x8f, 0xbb, 0x9d, 0x91,

```

```

    0xdf, 0x95, 0x7c, 0xff, 0x0a, 0x87, 0xaa, 0x45,
    0xde, 0xd3, 0xb9, 0x78, 0x12, 0xe8, 0x41, 0xc9,
    0xd4, 0x97, 0x7f, 0xd0, 0xe2, 0xd8, 0xea, 0xb4,
};
// For big-endian byte order
static CK_BYTE et_33[] = {
    0x18, 0x8f, 0x6c, 0xc3, 0x3f, 0x19, 0xbe, 0x1b,
    0x00, 0xa0, 0x67, 0x80, 0x90, 0xdb, 0xb2, 0x2b,
    0x5a, 0xaa, 0xb3, 0xec, 0x4e, 0x97, 0x5f, 0x6a,
    0xa5, 0xe5, 0x2b, 0xf3, 0x39, 0x57, 0x72, 0xeb
};
// For little-endian byte order
static CK_BYTE et_33_le[] = {
    0xbb, 0x9c, 0x11, 0x4a, 0x86, 0x5e, 0x3f, 0x1b,
    0xec, 0x2a, 0xd9, 0x68, 0x3a, 0x11, 0x14, 0xaf,
    0x8f, 0x6d, 0xed, 0x1a, 0x71, 0x4e, 0xa5, 0x41,
    0x01, 0x71, 0x3c, 0x9d, 0xc7, 0x89, 0x32, 0x1b,
};
CK_BYTE *state = NULL;
CK_ULONG state_len = 0;

key_template[0].pValue = keyval_32;
key_template[0].ulValueLen = sizeof(keyval_32);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4),
    value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

```

```
}

if (len != sizeof(et_32_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -2;
    goto end;
}
print_hex(value, len);
if (memcmp(value, et_32_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -3;
    goto end;
}
printf("Sign 32_le OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4),
    value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("Verify 32_le OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval_31;
key_template[0].ulValueLen = sizeof(keyval_31);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}
}
```

```
rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4),
    value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_31_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -4;
    goto end;
}
print_hex(value, len);
if (memcmp(value, et_31_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -5;
    goto end;
}
printf("Sign 31_le OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4),
    value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("Verify 31_le OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
```

```
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval_33;
key_template[0].ulValueLen = sizeof(keyval_33);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_33_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -8;
    goto end;
}
print_hex(value, len);
if (memcmp(value, et_33_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -9;
    goto end;
}

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
```

```
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

printf("keyval_33_hash multipart test\n");
key_template[0].pValue = keyval_33_hash;
key_template[0].ulValueLen = sizeof(keyval_33_hash);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

mechanism->pParameter = 0;
mechanism->ulParameterLen = 0;
rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_SignUpdate(sess, data4, 7);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignUpdate failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_GetOperationState(sess, NULL, &state_len);
if (rc != CKR_OK) {
```



```
fprintf(stderr ,
    "C_GetOperationState failed: 0x%x\n" , rc);
goto end;
}
state = (CK_BYTE *)malloc(state_len);
rc = funcs->C_GetOperationState(sess , state , &state_len);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "C_GetOperationState failed: 0x%x\n" , rc);
    goto end;
}

rc = funcs->C_SetOperationState(sess , state , state_len ,
    CK_INVALID_HANDLE, keyh);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "C_SetOperationState failed: 0x%x\n" , rc);
    goto end;
}

rc = funcs->C_SignUpdate(sess , data4+7, sizeof(data4)-7);
if (rc != CKR_OK) {
    fprintf(stderr , "C_SignUpdate failed: 0x%x\n" , rc);
    goto end;
}

rc = funcs->C_SignFinal(sess , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr , "C_SignFinal failed: 0x%x\n" , rc);
    goto end;
}

if (len != sizeof(et_33_le)) {
    fprintf(stderr , "Invalid length: %d\n" , len);
    rc = -6;
    goto end;
}
print_hex(value , len);
if (memcmp(value , et_33_le , len) != 0) {
    fprintf(stderr , "Invalid value\n");
    rc = -7;
    goto end;
}
printf("Sign 33_le multipart OK\n");
```

```
mechanism->pParameter = 0;
mechanism->ulParameterLen = 0;
rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SetOperationState(sess, state, state_len,
    CK_INVALID_HANDLE, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SetOperationState failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignUpdate(sess, data4+7, sizeof(data4)-7);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignUpdate failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignFinal(sess, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignFinal failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_33_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -6;
    goto end;
}
print_hex(value, len);
if (memcmp(value, et_33_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -7;
    goto end;
}
printf("Restored Sign 33_le multipart OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
```

```
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("Verify 33_le OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

printf("keyval_33 singlepart test\n");
key_template[0].pValue = keyval_33;
key_template[0].ulValueLen = sizeof(keyval_33);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_33_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -8;
}
```

```
    goto end;
}
print_hex(value, len);
if (memcmp(value, et_33_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -9;
    goto end;
}
printf("keyval_33 singlepart sign OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("keyval_33 singlepart verify OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

printf("keyval1 singlepart sign test\n");
key_template[0].pValue = keyval1;
key_template[0].ulValueLen = sizeof(keyval1);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}
```

```
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data1, strlen(data1), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et1)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -10;
    goto end;
}
print_hex(value, len);
if (memcmp(value, et1, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -11;
    goto end;
}
printf("keyval1 singlepart sign OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data1, strlen(data1), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("keyval1 singlepart verify OK\n");

printf("SUCCESS\n");
rc = CKR_OK;
end:
if (keyh != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess, keyh);
}

return rc;
```

```
}
```

ГОСТ Р34.11-2012, 512 бит

В данном примере демонстрируется генерация и проверка HMAC механизмом СКМ\GOSTR3411\12\512_HMAC.

Листинг 3.11: ckm_gostr3411_12_512_hmac.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost3411_12_512_hmac(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
}
```

```

rc = funcs->C_GetMechanismInfo( SlotId ,
    CKM_GOSTR3411_12_512_HMAC, &minfo);
if (rc != CKR_OK) {
    fprintf(stderr ,
"\n==== Mechanism CKM_GOSTR3411_12_512_HMAC "
"not supported =====\n");
} else {
    fprintf(stderr ,
"\n===== CKM_GOSTR3411_12_512_HMAC "
"test =====\n");
    rc = test_gost3411_12_512_hmac(hSession);
    if (rc != CKR_OK) {
        fprintf(stderr ,
            "ERROR CKM_GOSTR3411_12_512_HMAC failed, rc = 0x%x\n",
            rc);
    } else {
        fprintf(stderr ,
            "CKM_GOSTR3411_12_512_HMAC test passed.\n");
    }
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr ,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr , "C_CloseSession success\n");
}

out:
return rc;
}

CK_RV test_gost3411_12_512_hmac(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[512];
    CK_ULONG len;
    CK_MECHANISM mechanism_desc =
        {CKM_GOSTR3411_12_512_HMAC, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

```

```
CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
static CK_KEY_TYPE key_type = CKK_GENERIC_SECRET;
static CK_BBOOL ltrue = CK_TRUE;

static CK_BYTE keyval1 [] = {
    0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x0b, 0x0b, 0x0b, 0x0b,
};
static CK_BYTE data1 [] = "Hi There";
static CK_BYTE et1 [] = {
    0x86, 0xb6, 0xa0, 0x6b, 0xfa, 0x9f, 0x19, 0x74,
    0xaf, 0xf6, 0xcc, 0xd7, 0xfa, 0x3f, 0x83, 0x5f,
    0x0b, 0xd8, 0x50, 0x39, 0x5d, 0x60, 0x84, 0xef,
    0xc4, 0x7b, 0x9d, 0xda, 0x86, 0x1a, 0x2c, 0xdf,
    0x0d, 0xca, 0xf9, 0x59, 0x16, 0x07, 0x33, 0xd5,
    0x26, 0x9f, 0x65, 0x67, 0x96, 0x6d, 0xd7, 0xa9,
    0xf9, 0x32, 0xa7, 0x7c, 0xd6, 0xf0, 0x80, 0x01,
    0x2c, 0xd4, 0x76, 0xf1, 0xc2, 0xcc, 0x31, 0xbb,
};

static CK_BYTE keyval2 [] = { 'J', 'e', 'f', 'e' };
static CK_BYTE data2 [] = "what do ya want for nothing?";
static CK_BYTE et2 [] = {
    0x00, 0x61, 0x75, 0x04, 0x8e, 0x45, 0x72, 0xac,
    0x61, 0x85, 0xfa, 0xf5, 0xff, 0xae, 0x49, 0x62,
    0x97, 0xf9, 0x99, 0x3f, 0xf1, 0xab, 0xb7, 0x05,
    0xce, 0x43, 0xda, 0x09, 0x51, 0x56, 0x8d, 0x9a,
    0x00, 0x61, 0x75, 0x04, 0x8e, 0x45, 0x72, 0xac,
    0x61, 0x85, 0xfa, 0xf5, 0xff, 0xae, 0x49, 0x62,
    0x97, 0xf9, 0x99, 0x3f, 0xf1, 0xab, 0xb7, 0x05,
    0xce, 0x43, 0xda, 0x09, 0x51, 0x56, 0x8d, 0x9a,
};

static CK_BYTE keyval3 [] = {
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
    0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
};
```



```

0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa, 0xaa,
};
static CK_BYTE data3 [] =
    "Test Using Larger Than Block-Size Key - Hash Key First";
static CK_BYTE et3 [] = {
    0x91, 0x77, 0xa0, 0x5f, 0xee, 0xca, 0xfa, 0x5c,
    0xbb, 0x14, 0x8b, 0x17, 0x61, 0x63, 0x70, 0xb5,
    0xd5, 0x9b, 0x23, 0x4c, 0x56, 0x7e, 0x26, 0x0d,
    0xda, 0xe2, 0x67, 0x2a, 0x5f, 0xd3, 0x45, 0xfa,
    0x91, 0x77, 0xa0, 0x5f, 0xee, 0xca, 0xfa, 0x5c,
    0xbb, 0x14, 0x8b, 0x17, 0x61, 0x63, 0x70, 0xb5,
    0xd5, 0x9b, 0x23, 0x4c, 0x56, 0x7e, 0x26, 0x0d,
    0xda, 0xe2, 0x67, 0x2a, 0x5f, 0xd3, 0x45, 0xfa,
};
static CK_ATTRIBUTE key_template [] = {
    { CKA_VALUE, NULL_PTR, 0 },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_SIGN, &ltrue, sizeof(ltrue) },
    { CKA_VERIFY, &ltrue, sizeof(ltrue) }
};

static CK_BYTE data4 [] = {
    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
};
static CK_BYTE keyval_32 [] = {
    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
};
// For big-endian byte order
static CK_BYTE et_32 [] = {
    0x52, 0x67, 0xfb, 0xa5, 0xbf, 0x8c, 0x73, 0xaf,
    0x26, 0x2c, 0xc2, 0xa0, 0x45, 0x61, 0x86, 0x2e,
    0x28, 0xce, 0xd8, 0xe9, 0x6f, 0x0a, 0x85, 0xf8,
};

```

```

    0xf8, 0x8b, 0x59, 0x40, 0xb8, 0xf5, 0x4a, 0x56,
    0x52, 0x67, 0xfb, 0xa5, 0xbf, 0x8c, 0x73, 0xaf,
    0x26, 0x2c, 0xc2, 0xa0, 0x45, 0x61, 0x86, 0x2e,
    0x28, 0xce, 0xd8, 0xe9, 0x6f, 0x0a, 0x85, 0xf8,
    0xf8, 0x8b, 0x59, 0x40, 0xb8, 0xf5, 0x4a, 0x56
};
// For little-endian byte order
static CK_BYTE et_32_le[] = {
    0x35, 0xe2, 0x38, 0x6f, 0xe1, 0x5d, 0xcf, 0x24,
    0xac, 0x0d, 0xfd, 0x10, 0xfc, 0x8c, 0x12, 0xd4,
    0x5b, 0x0b, 0x45, 0xd4, 0xac, 0x19, 0xe4, 0x17,
    0xa4, 0xf3, 0x99, 0x08, 0xe1, 0x76, 0x5d, 0x54,
    0x63, 0xab, 0x51, 0x6c, 0xaa, 0x65, 0x5e, 0x8a,
    0xae, 0xa0, 0xe9, 0xc8, 0x32, 0x6b, 0x39, 0x8d,
    0x3a, 0x90, 0x3f, 0xb6, 0x06, 0x7e, 0x4f, 0x30,
    0xa9, 0x9c, 0x1d, 0xb3, 0xb7, 0xf2, 0x6e, 0xb1,
};
static CK_BYTE keyval_31[] = {
    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
};
// For big-endian byte order
static CK_BYTE et_31[] = {
    0x98, 0x24, 0x2e, 0xd7, 0x01, 0x14, 0xc6, 0x94,
    0xbf, 0xfd, 0x10, 0x7b, 0x2f, 0x7e, 0xc3, 0xc7,
    0x87, 0x14, 0x61, 0xaf, 0x7f, 0x39, 0xf4, 0x05,
    0xe7, 0x2b, 0xfb, 0x63, 0x36, 0x15, 0xf1, 0xce,
    0x98, 0x24, 0x2e, 0xd7, 0x01, 0x14, 0xc6, 0x94,
    0xbf, 0xfd, 0x10, 0x7b, 0x2f, 0x7e, 0xc3, 0xc7,
    0x87, 0x14, 0x61, 0xaf, 0x7f, 0x39, 0xf4, 0x05,
    0xe7, 0x2b, 0xfb, 0x63, 0x36, 0x15, 0xf1, 0xce
};
// For little-endian byte order
static CK_BYTE et_31_le[] = {
    0x78, 0xa5, 0xb1, 0xe4, 0xe8, 0x75, 0x59, 0x8b,
    0xc5, 0xf2, 0x2d, 0x4f, 0x92, 0x0b, 0xe3, 0x76,
    0xa9, 0xbc, 0x68, 0xc5, 0x59, 0xfa, 0xbd, 0xb1,
    0x67, 0xcd, 0x78, 0x4e, 0x75, 0x27, 0x62, 0x4d,
    0x59, 0x2d, 0x75, 0x2a, 0x99, 0xfd, 0x06, 0xfd,
    0x1d, 0xc0, 0x66, 0xff, 0x3a, 0xc8, 0x85, 0x3e,
    0x86, 0x71, 0x8b, 0x26, 0xd8, 0x0d, 0x85, 0xd6,
    0xda, 0xb3, 0x12, 0x2c, 0x47, 0xe4, 0x59, 0x0d,
};

```

```

};
static CK_BYTE keyval_33 [] = {
    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,
    0x0A
};
static CK_BYTE keyval_33_hash [] = {
    0x10, 0xff, 0xed, 0x16, 0x36, 0x4d, 0xa5, 0x3a,
    0x07, 0xc9, 0xba, 0x00, 0x0c, 0xb5, 0x55, 0x31,
    0xab, 0x53, 0xda, 0xf7, 0x1c, 0x1f, 0xfe, 0x31,
    0xad, 0x48, 0x81, 0xaf, 0xa1, 0x31, 0xae, 0xbe
};
// For big-endian byte order
static CK_BYTE et_33 [] = {
    0x18, 0x8f, 0x6c, 0xc3, 0x3f, 0x19, 0xbe, 0x1b,
    0x00, 0xa0, 0x67, 0x80, 0x90, 0xdb, 0xb2, 0x2b,
    0x5a, 0xaa, 0xb3, 0xec, 0x4e, 0x97, 0x5f, 0x6a,
    0xa5, 0xe5, 0x2b, 0xf3, 0x39, 0x57, 0x72, 0xeb,
    0x18, 0x8f, 0x6c, 0xc3, 0x3f, 0x19, 0xbe, 0x1b,
    0x00, 0xa0, 0x67, 0x80, 0x90, 0xdb, 0xb2, 0x2b,
    0x5a, 0xaa, 0xb3, 0xec, 0x4e, 0x97, 0x5f, 0x6a,
    0xa5, 0xe5, 0x2b, 0xf3, 0x39, 0x57, 0x72, 0xeb
};
// For little-endian byte order
static CK_BYTE et_33_le [] = {
    0x25, 0x39, 0x76, 0x8a, 0xa6, 0xa4, 0xae, 0xe2,
    0xa6, 0xb7, 0xb8, 0xb7, 0x9c, 0x85, 0x1f, 0xd1,
    0xce, 0x1a, 0xe7, 0x4b, 0x3c, 0xa6, 0xbc, 0x3d,
    0x6e, 0xeb, 0x5e, 0xef, 0x2b, 0x66, 0x09, 0xa1,
    0x5f, 0x22, 0xb6, 0x85, 0x9f, 0xf7, 0x53, 0x1a,
    0xdf, 0xd0, 0x6b, 0xbd, 0x5a, 0xe1, 0xc4, 0x43,
    0x5c, 0x8f, 0xdc, 0x81, 0x86, 0x0a, 0x03, 0x5a,
    0x52, 0x9c, 0x38, 0xda, 0x22, 0x16, 0xb7, 0x1e,
};
CK_BYTE *state = NULL;
CK_ULONG state_len = 0;

key_template[0].pValue = keyval_32;
key_template[0].ulValueLen = sizeof(keyval_32);
rc = funcs->C_CreateObject(sess,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE),

```

```
&keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4),
    value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_32_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -2;
    goto end;
}
print_hex(value, len);
if (memcmp(value, et_32_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -3;
    goto end;
}
printf("Sign 32_le OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4),
    value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
}
```

```
    goto end;
}
printf("Verify 32_le OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval_31;
key_template[0].ulValueLen = sizeof(keyval_31);
rc = funcs->C_CreateObject(sess,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data4, sizeof(data4),
    value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_31_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -4;
    goto end;
}
print_hex(value, len);
if (memcmp(value, et_31_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
}
```

```
rc = -5;
goto end;
}
printf("Sign 31_le OK\n");

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4),
    value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}
printf("Verify 31_le OK\n");

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval_33;
key_template[0].ulValueLen = sizeof(keyval_33);
rc = funcs->C_CreateObject(sess,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
```

```
rc = funcs->C_Sign(sess, data4, sizeof(data4), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et_33_le)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -8;
    goto end;
}
print_hex(value, len);
if (memcmp(value, et_33_le, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -9;
    goto end;
}

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data4, sizeof(data4), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

key_template[0].pValue = keyval1;
key_template[0].ulValueLen = sizeof(keyval1);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}
```

```
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

memset(value, 0, sizeof(value));
len = sizeof(value);
rc = funcs->C_Sign(sess, data1, strlen(data1), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

if (len != sizeof(et1)) {
    fprintf(stderr, "Invalid length: %d\n", len);
    rc = -10;
    goto end;
}

print_hex(value, len);
if (memcmp(value, et1, len) != 0) {
    fprintf(stderr, "Invalid value\n");
    rc = -11;
    goto end;
}

rc = funcs->C_VerifyInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Verify(sess, data1, strlen(data1), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    goto end;
}

printf("SUCCESS\n");
rc = CKR_OK;
end:
if (keyh != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess, keyh);
}
```



```
}  
  
return rc;  
}
```

3.3.6 Генерация ключевой пары, ЭЦП и ее проверка

ГОСТ Р34.11-94 и ГОСТ Р34.10-2001

В данном примере демонстрируется применение механизмов для генерации ключевой пары, генерации и проверки ЭЦП – СКМ_GOSTR3410_KEY_PAIR_GEN, СКМ_GOSTR3410 и СКМ_GOSTR3410_WITH_GOSTR3411. Первый механизм работает с готовым дайжестом подписываемых данных, а второй сам генерирует этот дайжест для подписи.

Листинг 3.12: ckm_gostr3410.c

```
#include "test_common.h"  
  
CK_RV test_crypto();  
CK_RV test_gost3410(CK_SESSION_HANDLE sess);  
  
int main(int argc, char *argv[]) {  
    CK_RV rc = CKR_OK;  
  
    rc = test_main(argc, argv);  
    if (rc != CKR_OK) {  
        fprintf(stderr, "test_main failed: 0x%x\n", rc);  
        return rc;  
    }  
    rc = test_crypto();  
    if (rc != CKR_OK) {  
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);  
        return rc;  
    }  
    return CKR_OK;  
}  
  
CK_RV test_crypto()  
{  
    CK_RV rc, ret;  
    CK_SESSION_HANDLE hSession;  
    CK_MECHANISM_INFO minfo;  
  
    rc = funcs->C_OpenSession(SlotId,  
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
```

```

    NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");

    // log in as normal user
    rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
        user_pin));
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_Login failed, rc = 0x%x\n", rc);
        goto out_close;
    }
    fprintf(stderr, "C_Login success\n");
    rc = funcs->C_GetMechanismInfo(SlotId, CKM_GOSTR3410, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n==== Mechanism CKM_GOSTR3410 not supported =====\n"
        );
    } else {
        fprintf(stderr,
            "\n===== CKM_GOSTR3410 test =====\n"
        );
        rc = test_gost3410(hSession);
        if (rc != CKR_OK) {
            fprintf(stderr,
                "ERROR CKM_GOSTR3410 failed, rc = 0x%x\n", rc);
        } else {
            fprintf(stderr, "CKM_GOSTR3410 test SUCCESS.\n");
        }
    }
}
out_close:
    if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
        fprintf(stderr,
            "Error: C_CloseSession failed with 0x%x\n", ret);
        rc = ret;
    }
    else {
        fprintf(stderr, "C_CloseSession success\n");
    }
}

```

```

out:
    return rc;
}

CK_RV test_gost3410(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[256];
    CK_ULONG len;
    CK_OBJECT_HANDLE pub_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE priv_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE new_key = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc = {CKM_GOSTR3410, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;
    CK_MECHANISM mechanism_gen_desc = {
        CKM_GOSTR3410_KEY_PAIR_GEN, NULL, 0};
    CK_MECHANISM_PTR mechanism_gen = &mechanism_gen_desc;

    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_OBJECT_CLASS oclass_pub = CKO_PUBLIC_KEY;
    static CK_OBJECT_CLASS oclass_priv = CKO_PRIVATE_KEY;
    static CK_KEY_TYPE key_type = CKK_GOSTR3410;

    static CK_BYTE gostR3410params[] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01 };
    static CK_BYTE gostR3411params[] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01 };
    static CK_BYTE data[] = {
        0xee, 0x8c, 0xd9, 0x40, 0x23, 0xdd, 0x9a, 0xf8,
        0x17, 0xdd, 0xe8, 0x1f, 0x02, 0x25, 0x5b, 0xb3,
        0xaa, 0x6b, 0xd3, 0x21, 0x09, 0x41, 0x95, 0x7e,
        0xea, 0x7e, 0x0e, 0x3c, 0x35, 0x9f, 0x04, 0x5a
    };
    static CK_BYTE pub_key_value[64];
    static CK_UTF8CHAR *label = "Yet Another Keypair";
    CK_ATTRIBUTE pub_template[] = {
        { CKA_TOKEN, &ltrue, sizeof(ltrue)},
        { CKA_GOSTR3410PARAMS, gostR3410params, sizeof(
            gostR3410params) },
        { CKA_GOSTR3411PARAMS, gostR3411params, sizeof(
            gostR3411params) },
        { CKA_VERIFY, &ltrue, sizeof(CK_BBOOL) },

```

```
};
CK_ATTRIBUTE priv_template [] = {
    { CKA_TOKEN, &ltrue, sizeof(ltrue) },
    { CKA_PRIVATE, &ltrue, sizeof(ltrue) },
    { CKA_SIGN, &ltrue, sizeof(CK_BBOOL) },
};

rc = funcs->C_GenerateKeyPair(sess, mechanism_gen,
    pub_template, sizeof(pub_template)/sizeof(CK_ATTRIBUTE),
    priv_template, sizeof(priv_template)/sizeof(CK_ATTRIBUTE),
    &pub_key, &priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GenerateKeyPair failed: 0x%x\n", rc);
    return rc;
}
fprintf(stderr, "C_GenerateKeyPair OK\n");

rc = funcs->C_SignInit(sess, mechanism, priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Sign(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    return rc;
}
fprintf(stderr, "C_Sign OK\n");

printf("SIGNATURE:\n");
print_hex(value, len);

rc = funcs->C_VerifyInit(sess, mechanism, pub_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_Verify(sess, data, sizeof(data), value, len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Verify failed: 0x%x\n", rc);
    return rc;
}
```

```
}
fprintf(stderr, "C_Verify OK\n");

value[5] ^= 0x08;
fprintf(stderr, "Signature changed for testing\n");
// Signature changed and not valid
rc = funcs->C_VerifyInit(sess, mechanism, pub_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_Verify(sess, data, sizeof(data), value, len);
if (rc == CKR_OK) {
    fprintf(stderr,
        "C_Verify not failed for bad signature: 0x%x\n", rc);
    return -1;
}
fprintf(stderr,
    "C_Verify failed for bad signature (0x%x) - it's OK\n", rc);

// Restore valid signature
value[5] ^= 0x08;
// Change data
data[7] ^= 0x3A;
fprintf(stderr, "Data changed for testing\n");
rc = funcs->C_VerifyInit(sess, mechanism, pub_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_Verify(sess, data, sizeof(data), value, len);
if (rc == CKR_OK) {
    fprintf(stderr,
        "C_Verify not failed for bad data: 0x%x\n", rc);
    return -1;
}
fprintf(stderr,
    "C_Verify failed for bad data (0x%x) - it's OK\n", rc);

rc = funcs->C_DestroyObject(sess, priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}
}
```

```
rc = funcs->C_DestroyObject(sess, pub_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

return rc;
}
```

ГОСТ Р34.11-2012 и ГОСТ Р34.10-2012, 256 бит

Поскольку размер дайджеста ГОСТ Р34.11-2012 совпадает с размером дайджеста ГОСТ Р34.11-94, а ГОСТ Р34.10-2001 и ГОСТ Р34.10-2012 (256 бит) – это фактически одинаковые алгоритмы, то генерации ключевой пары, генерация и проверка ЭЦП производятся в данном случае так же, как для ГОСТ Р34.11-94 и ГОСТ Р34.10-2001.

ГОСТ Р34.11-2012 и ГОСТ Р34.10-2012, 512 бит

В данном примере демонстрируется применение двух механизмов для генерации и проверки ЭЦП – СКМ_GOSTR3410_512 и СКМ_GOSTR3410_WITH_GOSTR3411_12_512. Первый механизм работает с готовым дайджестом подписываемых данных, а второй сам генерирует этот дайджест для подписи.

Листинг 3.13: ckm_gostr3410_512.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gostr3410_512(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
}
```

```
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession( SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");

    // log in as normal user
    rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
        user_pin));
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_Login failed, rc = 0x%x\n", rc);
        goto out_close;
    }
    fprintf(stderr, "C_Login success\n");
    rc = funcs->C_GetMechanismInfo(SlotId, CKM_GOSTR3410_512, &
        minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n==== Mechanism CKM_GOSTR3410_512 not supported
            =====\n");
    } else {
        fprintf(stderr,
            "\n==== CKM_GOSTR3410_512 test
            =====\n");
        rc = test_gostr3410_512(hSession);
        if (rc != CKR_OK) {
            fprintf(stderr,
                "ERROR CKM_GOSTR3410_512 failed, rc = 0x%x\n", rc);
        } else {
            fprintf(stderr, "CKM_GOSTR3410_512 test SUCCESS.\n");
        }
    }
}
```

```

    }
}
out_close:
    if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
        fprintf(stderr,
            "Error: C_CloseSession failed with 0x%x\n", ret);
        rc = ret;
    }
    else {
        fprintf(stderr, "C_CloseSession success\n");
    }

out:
    return rc;
}

CK_RV test_gostr3410_512(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[256];
    CK_ULONG len;
    CK_OBJECT_HANDLE pub_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE priv_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE new_key = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc = {CKM_GOSTR3410_512, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;
    CK_MECHANISM mechanism_gen_desc = {
        CKM_GOSTR3410_512_KEY_PAIR_GEN, NULL, 0};
    CK_MECHANISM_PTR mechanism_gen = &mechanism_gen_desc;

    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_OBJECT_CLASS oclass_pub = CKO_PUBLIC_KEY;
    static CK_OBJECT_CLASS oclass_priv = CKO_PRIVATE_KEY;
    static CK_KEY_TYPE key_type = CKK_GOSTR3410_512;

    static CK_BYTE tc26_decc_Test_der_oid[] =
        {0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01, 0x02, 0
        x00};
    static CK_BYTE tc26_decc_A_der_oid[] =
        {0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01, 0x02, 0
        x01};
    static CK_BYTE tc26_decc_B_der_oid[] =

```



```

{0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01, 0x02, 0
x02};
static CK_BYTE tc26_gost3411_2012_512_der_oid [] =
{0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03};

static CK_BYTE digest [] = {
    0x27, 0x55, 0x57, 0xa4, 0x7d, 0xcf, 0xcb, 0x82,
    0x35, 0xff, 0x02, 0x9b, 0x74, 0x83, 0x7f, 0x04,
    0x41, 0xef, 0xe4, 0x1a, 0xed, 0x12, 0xc2, 0x07,
    0x31, 0x3e, 0x83, 0xb2, 0x7a, 0xbd, 0x1e, 0x6a,
    0x9d, 0x89, 0x27, 0x13, 0xbc, 0x30, 0xa1, 0x6b,
    0xf9, 0x47, 0xd4, 0x6a, 0x59, 0xbb, 0xbb, 0x3a,
    0x33, 0xee, 0x33, 0x38, 0x53, 0x91, 0xc7, 0x36,
    0x75, 0xe7, 0xd0, 0xc3, 0x60, 0x21, 0x35, 0x40,
};
static CK_BYTE pub_key_value[128];
static CK_UTF8CHAR *label = "Yet Another Keypair";
CK_ATTRIBUTE pub_template [] = {
    { CKA_CLASS, &oclass_pub, sizeof(oclass_pub) },
    { CKA_TOKEN, &ltrue, sizeof(ltrue) },
    { CKA_GOSTR3410PARAMS,
    tc26_decc_A_der_oid, sizeof(tc26_decc_A_der_oid) },
    { CKA_GOSTR3411PARAMS,
    tc26_gost3411_2012_512_der_oid,
    sizeof(tc26_gost3411_2012_512_der_oid) },
    { CKA_VERIFY, &ltrue, sizeof(CK_BBOOL) },
};
CK_ATTRIBUTE priv_template [] = {
    { CKA_CLASS, &oclass_priv, sizeof(oclass_priv) },
    { CKA_TOKEN, &ltrue, sizeof(ltrue) },
    { CKA_PRIVATE, &ltrue, sizeof(ltrue) },
    { CKA_SIGN, &ltrue, sizeof(CK_BBOOL) },
};

rc = funcs->C_GenerateKeyPair(sess, mechanism_gen,
    pub_template, sizeof(pub_template)/sizeof(CK_ATTRIBUTE),
    priv_template, sizeof(priv_template)/sizeof(CK_ATTRIBUTE),
    &pub_key, &priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GenerateKeyPair failed: 0x%x\n", rc);
    return rc;
}
fprintf(stderr, "C_GenerateKeyPair OK\n");

```

```
rc = funcs->C_SignInit(sess , mechanism , priv_key);
if (rc != CKR_OK) {
    fprintf(stderr , "C_SignInit failed: 0x%x\n" , rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Sign(sess , digest , sizeof(digest) , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr , "C_Sign failed: 0x%x\n" , rc);
    return rc;
}
fprintf(stderr , "C_Sign OK\n");

printf("SIGNATURE:\n");
print_hex(value , len);

rc = funcs->C_VerifyInit(sess , mechanism , pub_key);
if (rc != CKR_OK) {
    fprintf(stderr , "C_VerifyInit failed: 0x%x\n" , rc);
    return rc;
}

rc = funcs->C_Verify(sess , digest , sizeof(digest) , value , len)
;
if (rc != CKR_OK) {
    fprintf(stderr , "C_Verify failed: 0x%x\n" , rc);
    return rc;
}
fprintf(stderr , "C_Verify OK\n");

value[5] ^= 0x08;
fprintf(stderr , "Signature changed for testing\n");
// Signature changed and not valid
rc = funcs->C_VerifyInit(sess , mechanism , pub_key);
if (rc != CKR_OK) {
    fprintf(stderr , "C_VerifyInit failed: 0x%x\n" , rc);
    return rc;
}
rc = funcs->C_Verify(sess , digest , sizeof(digest) , value , len)
;
if (rc == CKR_OK) {
    fprintf(stderr ,
        "C_Verify not failed for bad signature: 0x%x\n" , rc);
```

```
    return -1;
}
fprintf(stderr,
    "C_Verify failed for bad signature (0x%x) - it's OK\n", rc);

    // Restore valid signature
value[5] ^= 0x08;
// Change data
digest[7] ^= 0x3A;
fprintf(stderr, "Data changed for testing\n");
rc = funcs->C_VerifyInit(sess, mechanism, pub_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_VerifyInit failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_Verify(sess, digest, sizeof(digest), value, len)
;
if (rc == CKR_OK) {
    fprintf(stderr,
        "C_Verify not failed for bad data: 0x%x\n", rc);
    return -1;
}
fprintf(stderr,
    "C_Verify failed for bad data (0x%x) - it's OK\n", rc);

rc = funcs->C_DestroyObject(sess, priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess, pub_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

return rc;
}
```

3.3.7 Генерация ключей согласования

В данных примерах в двух сессиях сначала генерируются ключевые пары, а затем перекрестно генерируются и сравниваются ключи согласования по алгоритму VKO.

ГОСТ Р34.10-2001

Листинг 3.14: ckm_gostr3410_key_derive.c

```
#include "test_common.h"

int main(int argc, char* argv[])
{
    CK_RV rvResult;
#ifdef WIN32
    HMODULE hPkcsLib = NULL;
    HMODULE hPkcsLib2 = NULL;
#else
    void *hPkcsLib = NULL;
    void *hPkcsLib2 = NULL;
#endif
    CK_C_GetFunctionList pcGetFunctionList = 0;
    CK_C_GetFunctionList pcGetFunctionList2 = 0;
    CK_FUNCTION_LIST_PTR Pkcs11FuncList = NULL;
    CK_FUNCTION_LIST_PTR Pkcs11FuncList2 = NULL;
    CK_SLOT_ID_PTR pSlotList = NULL;
    CK_SLOT_ID_PTR pSlotList2 = NULL;
    CK_SLOT_ID SlotId;
    CK_SLOT_ID SlotId2;
    CK_ULONG ulSlotCount;
    CK_ULONG ulSlotCount2;
    CK_SLOT_INFO SlotInfo;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_HANDLE hSession2;

    CK_UTF8CHAR_PTR pcUserPIN = (CK_UTF8CHAR_PTR)"01234567"
    ;
    CK_ULONG ulPinLength = 8; // PIN length
    CK_UTF8CHAR_PTR pcUserPIN2 = (CK_UTF8CHAR_PTR)"01234567"
    ;
    CK_ULONG ulPinLength2 = 8; // PIN length

    CK_BBOOL blTrue = CK_TRUE,
        blFalse = CK_FALSE;
    CK_ULONG ulKeyType_Gost2001 = CKK_GOSTR3410,
```

```

        ulKeyType_Gost28147 = CKK_GOST28147,
        ulClass_PubKey = CKO_PUBLIC_KEY,
        ulClass_PriKey = CKO_PRIVATE_KEY,
        ulClass_SecKey = CKO_SECRET_KEY;
// PAR ECC XchA OID
CK_BYTE gostR3410params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x24, 0x00
};
// PAR HASH 1 OID
CK_BYTE gostR3411params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
};
// PAR CIPHER A OID
CK_BYTE gost28147params_A [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
};
// PAR CIPHER B OID
CK_BYTE gost28147params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02
};

// ltemplate for GOST R 34.10–2001 public key
CK_ATTRIBUTE caGOST_PublicKeyTemplate [] =
{
    {CKA_TOKEN,          &blTrue,          sizeof(CK_BBOOL)
},
    {CKA_PRIVATE,       &blFalse,         sizeof(CK_BBOOL)
},
    {CKA_GOSTR3410_PARAMS, gostR3410params, sizeof(
gostR3410params)},
    {CKA_GOSTR3411_PARAMS, gostR3411params, sizeof(
gostR3411params)},
    {CKA_GOST28147_PARAMS, gost28147params_A, sizeof(
gost28147params_A)},
};

// ltemplate for GOST R 34.10–2001 private key
CK_ATTRIBUTE caGOST_PrivateKeyTemplate [] =
{
    {CKA_TOKEN,          &blTrue,          sizeof(CK_BBOOL)
},
    {CKA_PRIVATE,       &blTrue,          sizeof(CK_BBOOL)
},
    {CKA_DERIVE,        &blTrue,          sizeof(CK_BBOOL)},
};

```

```

};
// ltemplate for derive key
CK_ATTRIBUTE      caDeriveKey [] =
{
  {CKA_CLASS,      &ulClass_SecKey ,   sizeof(CK_ULONG) },
  {CKA_KEY_TYPE,  &ulKeyType_Gost28147, sizeof(CK_ULONG) },
  {CKA_TOKEN,     &blFalse ,          sizeof(CK_BBOOL) },
  {CKA_SENSITIVE, &blFalse ,          sizeof(CK_BBOOL) },
  {CKA_EXTRACTABLE, &blTrue ,         sizeof(CK_BBOOL) },
  {CKA_ENCRYPT,    &blTrue ,           sizeof(CK_BBOOL) },
  {CKA_DECRYPT,    &blTrue ,           sizeof(CK_BBOOL) },
  {CKA_GOST28147PARAMS, gost28147params_A , sizeof(
gost28147params_A ) },
};

CK_ULONG
  ulPubKeyCount =
  sizeof(caGOST_PublicKeyTemplate) / sizeof(CK_ATTRIBUTE) ,
  ulPriKeyCount =
  sizeof(caGOST_PrivateKeyTemplate) / sizeof(CK_ATTRIBUTE) ,
  ulDeriveKeyCount =
  sizeof(caDeriveKey) / sizeof(CK_ATTRIBUTE) ;

CK_OBJECT_HANDLE
  hSendPubKey = 0, // handle to public key of
sender
  hSendPriKey = 0, // handle to private key of
sender
  hRecpPubKey = 0, // handle to public key of
recipient
  hRecpPriKey = 0, // handle to private key of
recipient
  hSendDH_Key = 0, // Diffy–Hellman key of the
sender
  hRecpDH_Key = 0; // Diffy–Hellman key of the
recipient
CK_MECHANISM
  cmKeyGenMechanism, // mechanism for key pair
generation
  cmDeriveMechanism; // mechanism for key
derivation

CK_ULONG

```

```

        i;

CK_BYTE_PTR
    pbSecretKeyParam = NULL,
    pbSend_PubKeyValue = NULL,
    pbRecp_PubKeyValue = NULL,
    pbSend_PriKeyValue = NULL,
    pbRecp_PriKeyValue = NULL,
    pbSendDH_Key_Value = NULL,
    pbRecpDH_Key_Value = NULL;

CK_ATTRIBUTE
    caSecretKeyParam =
        {CKA_GOST28147PARAMS, pbSecretKeyParam, 0},
    caSend_PubKeyValue = {CKA_VALUE,
    pbSend_PubKeyValue, 0},
    caSend_PriKeyValue = {CKA_VALUE,
    pbSend_PriKeyValue, 0},
    caRecp_PubKeyValue = {CKA_VALUE,
    pbRecp_PubKeyValue, 0},
    caRecp_PriKeyValue = {CKA_VALUE,
    pbRecp_PriKeyValue, 0},
    SendDH_Key_Value = {CKA_VALUE, pbSendDH_Key_Value, 0},
    RecpDH_Key_Value = {CKA_VALUE, pbRecpDH_Key_Value, 0};

// parameters for derivation mechanism
CK_GOSTR3410_DERIVE_PARAMS_PTR DeriveParams;
// UKM must be non-zero by RFC 4357
CK_ULONG ulUKMLen = 8;
    CK_BYTE fixed_ukm[8] = {
        0x9D, 0x23, 0x98, 0xC0, 0x12, 0x31, 0x2A, 0x8E
    };
char *TextBlock =
"This text block will be encrypted and the cipher text will be
    decrypted.";
CK_ULONG    ulDataSize = 0;           // size of text data

CK_BYTE_PTR pbCipherText = NULL;     // encrypted data
CK_ULONG    ulCipherSize = 0;        // size of encrypted
data
CK_BYTE_PTR pbDecryptedText = NULL;   // decrypted data
CK_ULONG    ulDecryptedDataSize = 0; // size of decrypted
data
CK_CHAR *api_path = PKCS11_API_PATH;

```

```

CK_UTF8CHAR *user_pin = "01234567";
CK_ULONG slot_num = 0;
CK_CHAR *api_path2 = PKCS11_API_PATH;
CK_UTF8CHAR *user_pin2 = "01234567";
CK_ULONG slot_num2 = 0;
SYSTEMTIME      t1, t2;
CK_ULONG        diff/*, min_time, max_time, avg_time*/;

printf("Starting CKM_GOSTR3410_KEY_DERIVE test\n");

for (i=1; i<(CK_ULONG)argc; i++) {
    if (strcmp("-api", argv[i]) == 0) {
        ++i;
        api_path = argv[i];
    } else if (strcmp("-slot", argv[i]) == 0) {
        ++i;
        slot_num = atoi(argv[i]);
    } else if (strcmp("-user_pin", argv[i]) == 0) {
        ++i;
        user_pin = argv[i];
    }
}

api_path2 = api_path;
user_pin2 = user_pin;
slot_num2 = slot_num;

#ifdef WIN32
    hPkcsLib = LoadLibrary(api_path);
#else
    hPkcsLib = dlopen(api_path, RILD_NOW);
#endif
    if ( hPkcsLib == NULL ) {
        printf(
            "Can't load PKCS#11 API library. "
            "Check API library path.\n");
#ifdef WIN32
        printf("dlerror: %s\n", dlerror());
#endif
    }
    return -1;
}
#ifdef WIN32
    pcGetFunctionList =
        (CK_C_GetFunctionList)GetProcAddress(

```



```
    hPkcsLib, "C_GetFunctionList");
#else
    pcGetFunctionList =
        (CK_C_GetFunctionList) dlsym(
            hPkcsLib, "C_GetFunctionList");
#endif

    // get PKCS #11 function list
    rvResult = pcGetFunctionList(&Pkcs11FuncList);
    printf("Load PKCS #11 function list result: 0x%x\n", rvResult);
    if (rvResult != CKR_OK) return rvResult;

    // initialize Cryptoki
    rvResult = Pkcs11FuncList->C_Initialize(NULL);
    printf("Initialize Cryptoki result: 0x%x\n", rvResult);
    if (rvResult != CKR_OK) return rvResult;

    // get slot list
    rvResult = Pkcs11FuncList->C_GetSlotList(CK_FALSE, NULL, &
        ulSlotCount);
    printf("Get slot list result: 0x%x\n", rvResult);
    if (rvResult != CKR_OK) return rvResult;

    if (ulSlotCount > 0)
    {
        // allocate memory for slot list
        pSlotList = (CK_SLOT_ID_PTR) malloc(
            ulSlotCount * sizeof(CK_SLOT_ID));

        rvResult =
            Pkcs11FuncList->C_GetSlotList(
                CK_FALSE, pSlotList, &ulSlotCount);

        if (rvResult != CKR_OK) return rvResult;
        printf("Slot count: %d\n", ulSlotCount);
    }
    else return -3;

    // get information about sender and recipient slots.
    for(i=0; i<ulSlotCount; ++i)
    {
        rvResult = Pkcs11FuncList->C_GetSlotInfo(pSlotList[i], &
            SlotInfo);
        if (rvResult == CKR_OK)
```

```
{ // if a token is present in this slot
    if ((SlotInfo.flags & CKF_TOKEN_PRESENT) ==
        CKF_TOKEN_PRESENT)
    {
        SlotId = pSlotList[i];
        break;
    }
}
}
if (i >= ulSlotCount) {
    printf("No slots with token present\n");
    return -3;
}

printf("Slot ID: %d\n", SlotId);

// open session for slot with ID = SlotId[0]
rvResult = Pkcs11FuncList->C_OpenSession(SlotId,
    (CKF_SERIAL_SESSION | CKF_RW_SESSION),
    NULL,
    0,
    &hSession);
printf("Open session result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// user login
rvResult = Pkcs11FuncList->C_Login(hSession,
    CKU_USER,
    user_pin,
    strlen(user_pin));
printf("Login result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;
// load library PKCS #11
#ifdef WIN32
    hPkcsLib2 = LoadLibrary(api_path2);
#else
    hPkcsLib2 = dlopen(api_path2, RTLD_NOW);
#endif
if ( hPkcsLib2 == NULL ) {
    printf("Can't load PKCS#11 API library. "
        "Check API library path.\n");
#ifdef WIN32
    printf("dlerror: %s\n", dlerror());
#endif
#endif
```

```
    return FALSE;
}
#ifdef WIN32
    pcGetFunctionList2 =
        (CK_C_GetFunctionList)GetProcAddress(
            hPkesLib2, "C_GetFunctionList");
#else
    pcGetFunctionList2 =
        (CK_C_GetFunctionList)dlsym(
            hPkesLib2, "C_GetFunctionList");
#endif

// get PKCS #11 function list
rvResult = pcGetFunctionList2(&Pkes11FuncList2);
printf("Load PKCS #11 function list result: 0x%x\n", rvResult)
;
if (rvResult != CKR_OK) return rvResult;

// initialize Cryptoki
rvResult = Pkes11FuncList2->C_Initialize(NULL);
printf("Initialize Cryptoki result: 0x%x\n", rvResult);
if (rvResult != CKR_OK &&
    rvResult != CKR_CRYPTOKI_ALREADY_INITIALIZED) return
rvResult;

// get slot list
rvResult = Pkes11FuncList2->C_GetSlotList(CK_FALSE, NULL, &
    ulSlotCount2);
printf("Get slot list result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

if (ulSlotCount2 > 0)
{ // allocate memory for slot list
    pSlotList2 =
        (CK_SLOT_ID_PTR) malloc(
            ulSlotCount2 * sizeof(CK_SLOT_ID));

    rvResult =
        Pkes11FuncList2->C_GetSlotList(CK_FALSE,
            pSlotList2, &ulSlotCount2);

    if (rvResult != CKR_OK) return rvResult;
    printf("Slot count: %d\n", ulSlotCount2);
}
```

```
else return -3;

// get information about sender and recipient slots.
for(i=0; i<ulSlotCount2; ++i)
{
    rvResult = Pkcs11FuncList2->C_GetSlotInfo(
        pSlotList2[i], &SlotInfo);
    if (rvResult == CKR_OK)
    { // if a token is present in this slot
        if ((SlotInfo.flags & CKF_TOKEN_PRESENT) ==
            CKF_TOKEN_PRESENT)
        {
            SlotId2 = pSlotList2[i];
            break;
        }
    }
}
if (i>=ulSlotCount2) {
    printf("No slots with token present\n");
    return -3;
}

printf("Slot ID 2: %d\n", SlotId2);
{
    CK_TOKEN_INFO tinfo;
    rvResult = Pkcs11FuncList2->C_GetTokenInfo(pSlotList2[i], &
tinfo);
    if (rvResult != CKR_OK) {
        printf("Can't get token info\n");
        return -4;
    }
}

// open session for slot with ID = SlotId2[0]
rvResult = Pkcs11FuncList2->C_OpenSession(SlotId2,
    (CKF_SERIAL_SESSION | CKF_RW_SESSION),
    NULL,
    0,
    &hSession2);
printf("Open session result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// User login may return CKR_USER_ALREADY_LOGGED_IN
// if the same token used for recipient.
```

```
rvResult = Pkcs11FuncList2->C_Login(hSession2 ,
    CKU_USER,
    user_pin2 ,
    strlen(user_pin2));
printf("Login result: 0x%x\n", rvResult);
if (rvResult != CKR_OK &&
    rvResult != CKR_USER_ALREADY_LOGGED_IN) return rvResult;

// generate sender key pair with GOST R 34.10-2001
printf("Generate key pair of sender\n");
printf("    mechanism type: CKM_GOSTR3410_KEY_PAIR_GEN\n");

cmKeyGenMechanism.mechanism = CKM_GOSTR3410_KEY_PAIR_GEN;
cmKeyGenMechanism.pParameter = NULL;
cmKeyGenMechanism.ulParameterLen = 0;
rvResult = Pkcs11FuncList->C_GenerateKeyPair(hSession ,
    &cmKeyGenMechanism ,
    caGOST_PublicKeyTemplate ,
    ulPubKeyCount ,
    caGOST_PrivateKeyTemplate ,
    ulPriKeyCount ,
    &hSendPubKey ,
    &hSendPriKey);
printf("    generate sender key pair: result: 0x%x\n", rvResult
);
if (rvResult != CKR_OK) return rvResult;

printf("    sender public key handle: %d\n",
    (unsigned long)hSendPubKey);
printf("    sender private key handle: %d\n",
    (unsigned long)hSendPriKey);

// generate recipient key pair with GOST R 34.10-2001
printf("Generate key pair of recipient\n");
printf("    mechanism type: CKM_GR3410_KEY_PAIR_GEN\n");

rvResult = Pkcs11FuncList2->C_GenerateKeyPair(hSession2 ,
    &cmKeyGenMechanism ,
    caGOST_PublicKeyTemplate ,
    ulPubKeyCount ,
    caGOST_PrivateKeyTemplate ,
    ulPriKeyCount ,
    &hRecpPubKey ,
    &hRecpPriKey);
```

```
printf("    generate recipient key pair: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    recipient public key handle: %d\n",
    (unsigned long)hRecpPubKey);
printf("    recipient private key handle: %d\n",
    (unsigned long)hRecpPriKey);

// get value of sender public key
printf("Get value of sender public key\n");

rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession,
    hSendPubKey,
    &caSend_PubKeyValue,
    1);
if (rvResult == CKR_OK)
{
    caSend_PubKeyValue.pValue =
        (CK_BYTE_PTR) malloc(caSend_PubKeyValue.ulValueLen);
    rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession,
        hSendPubKey,
        &caSend_PubKeyValue,
        1);
}
printf("    Get sender public key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

pbSend_PubKeyValue = (CK_BYTE_PTR) caSend_PubKeyValue.pValue;
printf("    Sender public key value: ");

for(i=0; i<caSend_PubKeyValue.ulValueLen; ++i)
    printf("%x ", pbSend_PubKeyValue[i]);
printf("\n");

// get value of recipient public key
printf("Get value of recipient public key\n");
rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
    hRecpPubKey,
    &caRecp_PubKeyValue,
    1);
if (rvResult == CKR_OK)
{
```

```

caRecp_PubKeyValue.pValue =
    (CK_BYTE_PTR) malloc(caRecp_PubKeyValue.ulValueLen);
rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
    hRecpPubKey,
    &caRecp_PubKeyValue,
    1);
}

printf("    Get recipient public key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    Recipient public key value: ");
pbRecp_PubKeyValue = (CK_BYTE_PTR) caRecp_PubKeyValue.pValue;

for (i=0; i<caRecp_PubKeyValue.ulValueLen; ++i)
    printf("%x ", pbRecp_PubKeyValue[i]);
printf("\n");

// fill parameters for derivation mechanism
DeriveParams = (CK_GOSTR3410_DERIVE_PARAMS_PTR)
    malloc(sizeof(CK_GOSTR3410_DERIVE_PARAMS));
DeriveParams->kdf = CKD_CPDIVERSIFY_KDF;
DeriveParams->pPublicData = (CK_BYTE_PTR) caRecp_PubKeyValue.
    pValue;
DeriveParams->ulPublicDataLen = caRecp_PubKeyValue.ulValueLen;
DeriveParams->pUKM = fixed_ukm;
DeriveParams->ulUKMLen = sizeof(fixed_ukm);

cmDeriveMechanism.mechanism = CKM_GOSTR3410_DERIVE;
cmDeriveMechanism.pParameter = DeriveParams;
cmDeriveMechanism.ulParameterLen = sizeof(
    CK_GOSTR3410_DERIVE_PARAMS);

printf("Derive Diffie-Hellman key\n");
printf("    derivation mechanism: CKM_GOSTR3410_DERIVE\n");

// derive Diffie-Hellman key of sender
printf("    derive Diffie-Hellman key for sender\n");
GetSystemTime(&t1);
rvResult = Pkcs11FuncList->C_DeriveKey(hSession,
    &cmDeriveMechanism,
    hSendPriKey,
    caDeriveKey,

```

```

        ulDeriveKeyCount ,
        &hSendDH_Key);
    GetSystemTime(&t2);
    diff = process_time(t1, t2);
    fprintf(stderr, "C_DeriveKey time: %ld msec\n", diff );
    printf("    derive key result: 0x%x\n", rvResult);
    if (rvResult != CKR_OK) return rvResult;

    printf("    sender's Diffie-Hellman key handle: %d\n",
        (unsigned long)hSendDH_Key);

    // get value of sender derived key
    printf("Get value of sender derived key\n");
    rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession ,
        hSendDH_Key,
        &SendDH_Key_Value,
        1);
    if (rvResult == CKR_OK)
    {
        SendDH_Key_Value.pValue =
            (CK_BYTE_PTR) malloc(SendDH_Key_Value.ulValueLen);
        rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession ,
            hSendDH_Key,
            &SendDH_Key_Value,
            1);
    }

    printf("    Get sender derived key value: result: 0x%x\n",
        rvResult);
    if (rvResult != CKR_OK) return rvResult;

    pbSendDH_Key_Value = (CK_BYTE_PTR) SendDH_Key_Value.pValue;

    printf("    Sender derived key value: ");

    for(i=0; i<SendDH_Key_Value.ulValueLen; ++i)
        printf("%x ", pbSendDH_Key_Value[i]);
    printf("\n");

    // derive Diffie-Hellman key of recipient
    printf("    derive Diffie-Hellman key for recipient\n");
    DeriveParams->pPublicData = (CK_BYTE_PTR) caSend_PubKeyValue.
        pValue;
    DeriveParams->ulPublicDataLen = caSend_PubKeyValue.ulValueLen;

```



```
rvResult = Pkcs11FuncList2->C_DeriveKey(hSession2 ,
    &cmDeriveMechanism ,
    hRecpPriKey ,
    caDeriveKey ,
    ulDeriveKeyCount ,
    &hRecpDH_Key);
printf("    derive key result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    recipient's Diffie-Hellman key handle: %d\n",
    (unsigned long)hRecpDH_Key);

// get value of recipient derived key
printf("Get value of recipient derived key\n");
rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2 ,
    hRecpDH_Key ,
    &RecpDH_Key_Value ,
    1);
if (rvResult == CKR_OK)
{
    RecpDH_Key_Value.pValue =
        (CK_BYTE_PTR) malloc(RecpDH_Key_Value.ulValueLen);
    rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2 ,
        hRecpDH_Key ,
        &RecpDH_Key_Value ,
        1);
}

printf("    Get recipient derived key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

pbRecpDH_Key_Value = (CK_BYTE_PTR) RecpDH_Key_Value.pValue;

printf("    Recipient derived key value: ");

for(i=0; i<RecpDH_Key_Value.ulValueLen; ++i)
    printf("%x ", pbRecpDH_Key_Value[i]);
printf("\n");

// Sender and recipient DH key values must be equal
if (RecpDH_Key_Value.ulValueLen != SendDH_Key_Value.ulValueLen
) {
```

```
    printf("Sender and recipient DH keys are of different length\n");
    return -1;
}
for (i=0; i<RecpDH_Key_Value.ulValueLen; ++i) {
    if (pbRecpDH_Key_Value[i] != pbSendDH_Key_Value[i]) {
        printf(
            "Sender and recipient KEK keys "
            "are different at position %d\n", i);
        return -1;
    }
}
printf("Sender and recipient KEK keys are equal\n");
printf("CKM_GOSTR3410_KEY_DERIVE test SUCCESS\n");
rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
    hSendPubKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
    hSendPriKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
    hRecpPubKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
    hRecpPriKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
    hSendDH_Key);
printf("    C_DestroyObject result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
    hRecpDH_Key);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult = Pkcs11FuncList->C_Logout(hSession);
printf("Sender logout result: 0x%x\n", rvResult);

// User logout may return CKR_USER_NOT_LOGGED_IN
// if the same token used for recipient.
rvResult = Pkcs11FuncList2->C_Logout(hSession2);
printf("Recipient logout result: 0x%x\n", rvResult);
// Close session
rvResult = Pkcs11FuncList->C_CloseSession(hSession);
```

```

printf("Sender close session result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList2->C_CloseSession(hSession2);
printf("Recipient close session result: 0x%x\n", rvResult);

return 0;
}

```

ГОСТ Р34.10-2012, 256 бит

Листинг 3.15: ckm_gostr3410_12_256_key_derive.c

```

#include "test_common.h"

int main(int argc, char* argv[])
{
    CK_RV rvResult;
#ifdef WIN32
    HMODULE hPkcsLib = NULL;
    HMODULE hPkcsLib2 = NULL;
#else
    void *hPkcsLib = NULL;
    void *hPkcsLib2 = NULL;
#endif
    CK_C_GetFunctionList pcGetFunctionList = 0;
    CK_C_GetFunctionList pcGetFunctionList2 = 0;
    CK_FUNCTION_LIST_PTR Pkcs11FuncList = NULL;
    CK_FUNCTION_LIST_PTR Pkcs11FuncList2 = NULL;
    CK_SLOT_ID_PTR pSlotList = NULL;
    CK_SLOT_ID_PTR pSlotList2 = NULL;
    CK_SLOT_ID SlotId;
    CK_SLOT_ID SlotId2;
    CK_ULONG ulSlotCount;
    CK_ULONG ulSlotCount2;
    CK_SLOT_INFO SlotInfo;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_HANDLE hSession2;

    CK_UTF8CHAR_PTR pcUserPIN = (CK_UTF8CHAR_PTR)"01234567"
    ;
    CK_ULONG ulPinLength = 8; // PIN length
    CK_UTF8CHAR_PTR pcUserPIN2 = (CK_UTF8CHAR_PTR)"01234567"
    ;
    CK_ULONG ulPinLength2 = 8; // PIN length

```

```

CK_BBOOL blTrue = CK_TRUE,
          blFalse = CK_FALSE;
CK_ULONG ulKeyType_Gost2001 = CKK_GOSTR3410,
          ulKeyType_Gost28147 = CKK_GOST28147,
          ulClass_PubKey = CKO_PUBLIC_KEY,
          ulClass_PriKey = CKO_PRIVATE_KEY,
          ulClass_SecKey = CKO_SECRET_KEY;
// PAR ECC XchA OID
CK_BYTE gostR3410params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x24, 0x00
};
// PAR 3411-2012-256 OID
CK_BYTE gostR3411params [] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02
};
// PAR CIPHER A OID
CK_BYTE gost28147params_A [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
};
// PAR CIPHER B OID
CK_BYTE gost28147params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02
};
// Когда закрытый ключ 256 бит, то только по значению
// атрибута CKA_GOSTR3411_PARAMS можно разобраться,
// какой алгоритм диверсификации нужно применять
// для результирующего ключа.
CK_ATTRIBUTE caGOST_PublicKeyTemplate [] =
{
    {CKA_TOKEN,          &blTrue,          sizeof(CK_BBOOL) },
    {CKA_PRIVATE,       &blFalse,         sizeof(CK_BBOOL) },
    {CKA_GOSTR3410_PARAMS,
     gostR3410params,   sizeof(gostR3410params) },
    {CKA_GOSTR3411_PARAMS,
     gostR3411params,   sizeof(gostR3411params) },
    {CKA_GOST28147_PARAMS,
     gost28147params_A, sizeof(gost28147params_A) },
};
CK_ATTRIBUTE caGOST_PrivateKeyTemplate [] =
{
    {CKA_TOKEN,          &blTrue,          sizeof(CK_BBOOL) },
    {CKA_PRIVATE,       &blTrue,           sizeof(CK_BBOOL) },
    {CKA_DERIVE,        &blTrue,           sizeof(CK_BBOOL) },
};

```

```

};
// ltemplate for derive key
CK_ATTRIBUTE      caDeriveKey [] =
{
  {CKA_CLASS,      &ulClass_SecKey,   sizeof(CK_ULONG) },
  {CKA_KEY_TYPE,   &ulKeyType_Gost28147, sizeof(CK_ULONG) },
  {CKA_TOKEN,      &blFalse,         sizeof(CK_BBOOL) },
  {CKA_SENSITIVE,  &blFalse,         sizeof(CK_BBOOL) },
  {CKA_EXTRACTABLE, &blTrue,         sizeof(CK_BBOOL) },
  {CKA_ENCRYPT,     &blTrue,          sizeof(CK_BBOOL) },
  {CKA_DECRYPT,     &blTrue,          sizeof(CK_BBOOL) },
  {CKA_GOST28147PARAMS,
   gost28147params_A, sizeof(gost28147params_A) },
};

CK_ULONG
  ulPubKeyCount =
  sizeof(caGOST_PublicKeyTemplate) / sizeof(CK_ATTRIBUTE) ,
  ulPriKeyCount =
  sizeof(caGOST_PrivateKeyTemplate) / sizeof(CK_ATTRIBUTE) ,
  ulDeriveKeyCount =
  sizeof(caDeriveKey) / sizeof(CK_ATTRIBUTE) ;

CK_OBJECT_HANDLE
  hSendPubKey = 0, // handle to public key of
sender
  hSendPriKey = 0, // handle to private key of
sender
  hRecpPubKey = 0, // handle to public key of
recipient
  hRecpPriKey = 0, // handle to private key of
recipient
  hSendDH_Key = 0, // Diffy–Hellman key of the
sender
  hRecpDH_Key = 0; // Diffy–Hellman key of the
recipient
CK_MECHANISM
  cmKeyGenMechanism, // mechanism for key pair
generation
  cmDeriveMechanism; // mechanism for key
derivation

CK_ULONG

```

```

        i;

CK_BYTE_PTR
    pbSecretKeyParam = NULL,
    pbSend_PubKeyValue = NULL,
    pbRecp_PubKeyValue = NULL,
    pbSend_PriKeyValue = NULL,
    pbRecp_PriKeyValue = NULL,
    pbSendDH_Key_Value = NULL,
    pbRecpDH_Key_Value = NULL;

CK_ATTRIBUTE
    caSecretKeyParam =
        {CKA_GOST28147PARAMS, pbSecretKeyParam, 0},
    caSend_PubKeyValue =
        {CKA_VALUE, pbSend_PubKeyValue, 0},
    caSend_PriKeyValue =
        {CKA_VALUE, pbSend_PriKeyValue, 0},
    caRecp_PubKeyValue =
        {CKA_VALUE, pbRecp_PubKeyValue, 0},
    caRecp_PriKeyValue =
        {CKA_VALUE, pbRecp_PriKeyValue, 0},
    SendDH_Key_Value = {CKA_VALUE, pbSendDH_Key_Value, 0},
    RecpDH_Key_Value = {CKA_VALUE, pbRecpDH_Key_Value, 0};

// parameters for derivation mechanism
CK_GOSTR3410_DERIVE_PARAMS_PTR DeriveParams;
// UKM must be non-zero by RFC 4357
CK_ULONG ulUKMLen = 8;
    CK_BYTE fixed_ukm[8] = {
        0x9D, 0x23, 0x98, 0xC0, 0x12, 0x31, 0x2A, 0x8E
    };
    char *TextBlock =
"This text block will be encrypted and the cipher text will be
    decrypted.";
    CK_ULONG ulDataSize = 0; // size of text data

    CK_BYTE_PTR pbCipherText = NULL; // encrypted data
    CK_ULONG ulCipherSize = 0; // size of encrypted
    data
    CK_BYTE_PTR pbDecryptedText = NULL; // decrypted data
    CK_ULONG ulDecryptedDataSize = 0; // size of decrypted
    data
    CK_CHAR *api_path = PKCS11_API_PATH;

```

```

CK_UTF8CHAR *user_pin = "01234567";
CK_ULONG slot_num = 0;
CK_CHAR *api_path2 = PKCS11_API_PATH;
CK_UTF8CHAR *user_pin2 = "01234567";
CK_ULONG slot_num2 = 0;
SYSTEMTIME      t1, t2;
CK_ULONG        diff/*, min_time, max_time, avg_time*/;

printf("Starting CKM_GOSTR3410_KEY_DERIVE (2012_256) test\n");

for (i=1; i<(CK_ULONG)argc; i++) {
    if (strcmp("-api", argv[i]) == 0) {
        ++i;
        api_path = argv[i];
    } else if (strcmp("-slot", argv[i]) == 0) {
        ++i;
        slot_num = atoi(argv[i]);
    } else if (strcmp("-user_pin", argv[i]) == 0) {
        ++i;
        user_pin = argv[i];
    }
}

api_path2 = api_path;
user_pin2 = user_pin;
slot_num2 = slot_num;

#ifdef WIN32
    hPkcsLib = LoadLibrary(api_path);
#else
    hPkcsLib = dlopen(api_path, RILD_NOW);
#endif
    if ( hPkcsLib == NULL ) {
        printf(
            "Can't load PKCS#11 API library. "
            "Check API library path.\n");
#ifdef WIN32
        printf("dlerror: %s\n", dlerror());
#endif
    }
    return -1;
}
#ifdef WIN32
    pcGetFunctionList =
        (CK_C_GetFunctionList)GetProcAddress(

```

```
        hPkcsLib, "C_GetFunctionList");
#else
    pcGetFunctionList =
        (CK_C_GetFunctionList) dlsym(
            hPkcsLib, "C_GetFunctionList");
#endif

    // get PKCS #11 function list
    rvResult = pcGetFunctionList(&Pkcs11FuncList);
    printf("Load PKCS #11 function list result: 0x%x\n", rvResult);
    if (rvResult != CKR_OK) return rvResult;

    // initialize Cryptoki
    rvResult = Pkcs11FuncList->C_Initialize(NULL);
    printf("Initialize Cryptoki result: 0x%x\n", rvResult);
    if (rvResult != CKR_OK) return rvResult;

    // get slot list
    rvResult =
        Pkcs11FuncList->C_GetSlotList(CK_FALSE, NULL, &ulSlotCount);
    printf("Get slot list result: 0x%x\n", rvResult);
    if (rvResult != CKR_OK) return rvResult;

    if (ulSlotCount > 0)
    {
        // allocate memory for slot list
        pSlotList = (CK_SLOT_ID_PTR) malloc(
            ulSlotCount * sizeof(CK_SLOT_ID));

        rvResult =
            Pkcs11FuncList->C_GetSlotList(
                CK_FALSE, pSlotList, &ulSlotCount);

        if (rvResult != CKR_OK) return rvResult;
        printf("Slot count: %d\n", ulSlotCount);
    }
    else return -3;

    // get information about sender and recipient slots.
    for (i=0; i<ulSlotCount; ++i)
    {
        rvResult =
            Pkcs11FuncList->C_GetSlotInfo(pSlotList[i], &SlotInfo);
        if (rvResult == CKR_OK)
```



```
{ // if a token is present in this slot
  if ((SlotInfo.flags & CKF_TOKEN_PRESENT) ==
      CKF_TOKEN_PRESENT)
  {
    SlotId = pSlotList[i];
    break;
  }
}
}
if (i >= ulSlotCount) {
  printf("No slots with token present\n");
  return -3;
}

printf("Slot ID: %d\n", SlotId);

// open session for slot with ID = SlotId[0]
rvResult = Pkcs11FuncList->C_OpenSession(SlotId,
      (CKF_SERIAL_SESSION | CKF_RW_SESSION),
      NULL,
      0,
      &hSession);
printf("Open session result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// user login
rvResult = Pkcs11FuncList->C_Login(hSession,
      CKU_USER,
      user_pin,
      strlen(user_pin));
printf("Login result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;
// load library PKCS #11
#ifdef WIN32
  hPkcsLib2 = LoadLibrary(api_path2);
#else
  hPkcsLib2 = dlopen(api_path2, RTLD_NOW);
#endif
if ( hPkcsLib2 == NULL ) {
  printf("Can't load PKCS#11 API library. "
      "Check API library path.\n");
#ifdef WIN32
  printf("dlerror: %s\n", dlerror());
#endif
#endif
```

```
    return FALSE;
}
#ifdef WIN32
pcGetFunctionList2 =
    (CK_C_GetFunctionList)GetProcAddress(
        hPkesLib2, "C_GetFunctionList");
#else
pcGetFunctionList2 =
    (CK_C_GetFunctionList)dlsym(
        hPkesLib2, "C_GetFunctionList");
#endif

// get PKCS #11 function list
rvResult = pcGetFunctionList2(&Pkes11FuncList2);
printf("Load PKCS #11 function list result: 0x%x\n", rvResult);
;
if (rvResult != CKR_OK) return rvResult;

// initialize Cryptoki
rvResult = Pkes11FuncList2->C_Initialize(NULL);
printf("Initialize Cryptoki result: 0x%x\n", rvResult);
if (rvResult != CKR_OK &&
    rvResult != CKR_CRYPTOKI_ALREADY_INITIALIZED)
    return rvResult;

// get slot list
rvResult =
    Pkes11FuncList2->C_GetSlotList(CK_FALSE, NULL, &ulSlotCount2
    );
printf("Get slot list result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

if (ulSlotCount2 > 0)
{ // allocate memory for slot list
    pSlotList2 =
        (CK_SLOT_ID_PTR) malloc(
            ulSlotCount2 * sizeof(CK_SLOT_ID));

    rvResult =
        Pkes11FuncList2->C_GetSlotList(CK_FALSE,
            pSlotList2, &ulSlotCount2);

    if (rvResult != CKR_OK) return rvResult;
    printf("Slot count: %d\n", ulSlotCount2);
}
```

```
}
else return -3;

// get information about sender and recipient slots.
for(i=0; i<ulSlotCount2; ++i)
{
    rvResult = Pkcs11FuncList2->C_GetSlotInfo(
        pSlotList2[i], &SlotInfo);
    if (rvResult == CKR_OK)
    { // if a token is present in this slot
        if ((SlotInfo.flags & CKF_TOKEN_PRESENT) ==
            CKF_TOKEN_PRESENT)
        {
            SlotId2 = pSlotList2[i];
            break;
        }
    }
}
if (i>=ulSlotCount2) {
    printf("No slots with token present\n");
    return -3;
}

printf("Slot ID 2: %d\n", SlotId2);
{
    CK_TOKEN_INFO tinfo;
    rvResult = Pkcs11FuncList2->C_GetTokenInfo(pSlotList2[i], &
tinfo);
    if (rvResult != CKR_OK) {
        printf("Can't get token info\n");
        return -4;
    }
}

// open session for slot with ID = SlotId2[0]
rvResult = Pkcs11FuncList2->C_OpenSession(SlotId2,
    (CKF_SERIAL_SESSION | CKF_RW_SESSION),
    NULL,
    0,
    &hSession2);
printf("Open session result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// User login may return CKR_USER_ALREADY_LOGGED_IN
```

```
// if the same token used for recipient.
rvResult = Pkcs11FuncList2->C_Login(hSession2 ,
    CKU_USER,
    user_pin2 ,
    strlen(user_pin2));
printf("Login result: 0x%x\n", rvResult);
if (rvResult != CKR_OK &&
    rvResult != CKR_USER_ALREADY_LOGGED_IN) return rvResult;

// generate sender key pair with GOST R 34.10-2001
printf("Generate key pair of sender\n");
printf("    mechanism type: CKM_GOSTR3410_KEY_PAIR_GEN\n");

cmKeyGenMechanism.mechanism = CKM_GOSTR3410_KEY_PAIR_GEN;
cmKeyGenMechanism.pParameter = NULL;
cmKeyGenMechanism.ulParameterLen = 0;
rvResult = Pkcs11FuncList->C_GenerateKeyPair(hSession ,
    &cmKeyGenMechanism ,
    caGOST_PublicKeyTemplate ,
    ulPubKeyCount ,
    caGOST_PrivateKeyTemplate ,
    ulPriKeyCount ,
    &hSendPubKey ,
    &hSendPriKey);
printf("    generate sender key pair: result: 0x%x\n", rvResult
);
if (rvResult != CKR_OK) return rvResult;

printf("    sender public key handle: %d\n",
    (unsigned long)hSendPubKey);
printf("    sender private key handle: %d\n",
    (unsigned long)hSendPriKey);

// generate recipient key pair with GOST R 34.10-2001
printf("Generate key pair of recipient\n");
printf("    mechanism type: CKM_GR3410_KEY_PAIR_GEN\n");

rvResult = Pkcs11FuncList2->C_GenerateKeyPair(hSession2 ,
    &cmKeyGenMechanism ,
    caGOST_PublicKeyTemplate ,
    ulPubKeyCount ,
    caGOST_PrivateKeyTemplate ,
    ulPriKeyCount ,
    &hRecpPubKey ,
```

```
&hRecpPriKey);
printf("    generate recipient key pair: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    recipient public key handle: %d\n",
    (unsigned long)hRecpPubKey);
printf("    recipient private key handle: %d\n",
    (unsigned long)hRecpPriKey);

// get value of sender public key
printf("Get value of sender public key\n");

rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession,
    hSendPubKey,
    &caSend_PubKeyValue,
    1);
if (rvResult == CKR_OK)
{
    caSend_PubKeyValue.pValue =
        (CK_BYTE_PTR) malloc(caSend_PubKeyValue.ulValueLen);
    rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession,
        hSendPubKey,
        &caSend_PubKeyValue,
        1);
}
printf("    Get sender public key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

pbSend_PubKeyValue = (CK_BYTE_PTR) caSend_PubKeyValue.pValue;
printf("    Sender public key value: ");

for(i=0; i<caSend_PubKeyValue.ulValueLen; ++i)
    printf("%x ", pbSend_PubKeyValue[i]);
printf("\n");

// get value of recipient public key
printf("Get value of recipient public key\n");
rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
    hRecpPubKey,
    &caRecp_PubKeyValue,
    1);
if (rvResult == CKR_OK)
```

```

{
    caRecp_PubKeyValue.pValue =
        (CK_BYTE_PTR) malloc(caRecp_PubKeyValue.ulValueLen);
    rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
        hRecpPubKey,
        &caRecp_PubKeyValue,
        1);
}

printf("    Get recipient public key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    Recipient public key value: ");
pbRecp_PubKeyValue = (CK_BYTE_PTR) caRecp_PubKeyValue.pValue;

for (i=0; i<caRecp_PubKeyValue.ulValueLen; ++i)
    printf("%x ", pbRecp_PubKeyValue[i]);
printf("\n");

// fill parameters for derivation mechanism
DeriveParams = (CK_GOSTR3410_DERIVE_PARAMS_PTR)
    malloc(sizeof(CK_GOSTR3410_DERIVE_PARAMS));
DeriveParams->kdf = CKD_CPDIVERSIFY_KDF;
DeriveParams->pPublicData = (CK_BYTE_PTR) caRecp_PubKeyValue.
    pValue;
DeriveParams->ulPublicDataLen = caRecp_PubKeyValue.ulValueLen;
DeriveParams->pUKM = fixed_ukm;
DeriveParams->ulUKMLen = sizeof(fixed_ukm);

cmDeriveMechanism.mechanism = CKM_GOSTR3410_DERIVE;
cmDeriveMechanism.pParameter = DeriveParams;
cmDeriveMechanism.ulParameterLen = sizeof(
    CK_GOSTR3410_DERIVE_PARAMS);

printf("Derive Diffie-Hellman key\n");
printf("    derivation mechanism: CKM_GOSTR3410_DERIVE\n");

// derive Diffie-Hellman key of sender
printf("    derive Diffie-Hellman key for sender\n");
    GetSystemTime(&t1);
rvResult = Pkcs11FuncList->C_DeriveKey(hSession,
    &cmDeriveMechanism,
    hSendPriKey,

```

```
        caDeriveKey ,
        ulDeriveKeyCount ,
        &hSendDH_Key);
    GetSystemTime(&t2);
    diff = process_time(t1, t2);
    fprintf(stderr, "C_DeriveKey time: %ld msec\n", diff );
    printf("    derive key result: 0x%x\n", rvResult);
    if (rvResult != CKR_OK) return rvResult;

    printf("    sender's Diffie-Hellman key handle: %d\n",
        (unsigned long)hSendDH_Key);

    // get value of sender derived key
    printf("Get value of sender derived key\n");
    rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession ,
        hSendDH_Key,
        &SendDH_Key_Value,
        1);
    if (rvResult == CKR_OK)
    {
        SendDH_Key_Value.pValue =
            (CK_BYTE_PTR) malloc(SendDH_Key_Value.ulValueLen);
        rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession ,
            hSendDH_Key,
            &SendDH_Key_Value,
            1);
    }

    printf("    Get sender derived key value: result: 0x%x\n",
        rvResult);
    if (rvResult != CKR_OK) return rvResult;

    pbSendDH_Key_Value = (CK_BYTE_PTR) SendDH_Key_Value.pValue;

    printf("    Sender derived key value: ");

    for(i=0; i<SendDH_Key_Value.ulValueLen; ++i)
        printf("%x ", pbSendDH_Key_Value[i]);
    printf("\n");

    // derive Diffie-Hellman key of recipient
    printf("    derive Diffie-Hellman key for recipient\n");
    DeriveParams->pPublicData = (CK_BYTE_PTR) caSend_PubKeyValue.
        pValue;
```

```
DeriveParams->ulPublicDataLen = caSend_PubKeyValue.ulValueLen;

rvResult = Pkcs11FuncList2->C_DeriveKey(hSession2,
    &cmDeriveMechanism,
    hRecpPriKey,
    caDeriveKey,
    ulDeriveKeyCount,
    &hRecpDH_Key);
printf("    derive key result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    recipient's Diffie-Hellman key handle: %d\n",
    (unsigned long)hRecpDH_Key);

// get value of recipient derived key
printf("Get value of recipient derived key\n");
rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
    hRecpDH_Key,
    &RecpDH_Key_Value,
    1);
if (rvResult == CKR_OK)
{
    RecpDH_Key_Value.pValue =
        (CK_BYTE_PTR) malloc(RecpDH_Key_Value.ulValueLen);
    rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
        hRecpDH_Key,
        &RecpDH_Key_Value,
        1);
}

printf("    Get recipient derived key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

pbRecpDH_Key_Value = (CK_BYTE_PTR) RecpDH_Key_Value.pValue;

printf("    Recipient derived key value: ");

for(i=0; i<RecpDH_Key_Value.ulValueLen; ++i)
    printf("%x ", pbRecpDH_Key_Value[i]);
printf("\n");

// Sender and recipient DH key values must be equal
```



```
if (RecvDH_Key_Value.ulValueLen != SendDH_Key_Value.ulValueLen
) {
    printf("Sender and recipient DH keys are of different length
\n");
    return -1;
}
for(i=0; i<RecvDH_Key_Value.ulValueLen; ++i) {
    if (pbRecvDH_Key_Value[i] != pbSendDH_Key_Value[i]) {
        printf(
            "Sender and recipient KEK keys "
            "are different at position %d\n", i);
        return -1;
    }
}
printf("Sender and recipient KEK keys are equal\n");
printf("CKM_GOSTR3410_KEY_DERIVE test SUCCESS\n");
rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
    hSendPubKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
    hSendPriKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
    hRecvPubKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
    hRecvPriKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
    hSendDH_Key);
printf("    C_DestroyObject result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
    hRecvDH_Key);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult = Pkcs11FuncList->C_Logout(hSession);
printf("Sender logout result: 0x%x\n", rvResult);

// User logout may return CKR_USER_NOT_LOGGED_IN
// if the same token used for recipient.
rvResult = Pkcs11FuncList2->C_Logout(hSession2);
printf("Recipient logout result: 0x%x\n", rvResult);
```

```

// Close session
rvResult = Pkcs11FuncList->C_CloseSession(hSession);
printf("Sender close session result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList2->C_CloseSession(hSession2);
printf("Recipient close session result: 0x%x\n", rvResult);

printf("CKM_GOSTR3410_KEY_DERIVE (2012_256) test SUCCESS\n");
return 0;
}

```

ГОСТ Р34.10-2012, 512 бит

Листинг 3.16: ckm_gostr3410_12_512_key_derive.c

```

#include "test_common.h"

int main(int argc, char* argv[])
{
    CK_RV rvResult;
#ifdef WIN32
    HMODULE hPkcsLib = NULL;
    HMODULE hPkcsLib2 = NULL;
#else
    void *hPkcsLib = NULL;
    void *hPkcsLib2 = NULL;
#endif
    CK_C_GetFunctionList pcGetFunctionList = 0;
    CK_C_GetFunctionList pcGetFunctionList2 = 0;
    CK_FUNCTION_LIST_PTR Pkcs11FuncList = NULL;
    CK_FUNCTION_LIST_PTR Pkcs11FuncList2 = NULL;
    CK_SLOT_ID_PTR pSlotList = NULL;
    CK_SLOT_ID_PTR pSlotList2 = NULL;
    CK_SLOT_ID SlotId;
    CK_SLOT_ID SlotId2;
    CK_ULONG ulSlotCount;
    CK_ULONG ulSlotCount2;
    CK_SLOT_INFO SlotInfo;
    CK_SESSION_HANDLE hSession;
    CK_SESSION_HANDLE hSession2;

    CK_UTF8CHAR_PTR pcUserPIN = (CK_UTF8CHAR_PTR)"01234567"
;
    CK_ULONG ulPinLength = 8; // PIN length

```

```

    CK_UTF8CHAR_PTR      pcUserPIN2 = (CK_UTF8CHAR_PTR)"01234567
";
    CK_ULONG              ulPinLength2 = 8;           // PIN length

    CK_BBOOL blTrue = CK_TRUE,
              blFalse = CK_FALSE;
    CK_ULONG ulKeyType_Gost2001 = CKK_GOSTR3410,
              ulKeyType_Gost28147 = CKK_GOST28147,
              ulClass_PubKey = CKO_PUBLIC_KEY,
              ulClass_PriKey = CKO_PRIVATE_KEY,
              ulClass_SecKey = CKO_SECRET_KEY;
    // PAR 512 A OID
    CK_BYTE gostR3410params [] = {
        0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x2, 0x01, 0x02, 0
        x01
    };
    // PAR 3411-2012-512 OID
    CK_BYTE gostR3411params [] = {
        0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03
    };
    // PAR CIPHER A OID
    CK_BYTE gost28147params_A [] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
    };
    // PAR CIPHER B OID
    CK_BYTE gost28147params [] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02
    };
    CK_ATTRIBUTE caGOST_PublicKeyTemplate [] =
    {
        {CKA_TOKEN,          &blTrue,          sizeof(CK_BBOOL)
        },
        {CKA_PRIVATE,       &blFalse,         sizeof(CK_BBOOL)
        },
        {CKA_GOSTR3410_PARAMS, gostR3410params, sizeof(
        gostR3410params)},
        {CKA_GOSTR3411_PARAMS, gostR3411params, sizeof(
        gostR3411params)},
        {CKA_GOST28147_PARAMS, gost28147params_A, sizeof(
        gost28147params_A)},
    };
    CK_ATTRIBUTE caGOST_PrivateKeyTemplate [] =
    {

```

```

    {CKA_TOKEN,          &blTrue,          sizeof(CK_BBOOL)
    },
    {CKA_PRIVATE,       &blTrue,          sizeof(CK_BBOOL)
    },
    {CKA_DERIVE,        &blTrue,          sizeof(CK_BBOOL)},
    };

// ltemplate for derive key
CK_ATTRIBUTE          caDeriveKey [] =
{
    {CKA_CLASS,         &ulClass_SecKey,   sizeof(CK_ULONG)},
    {CKA_KEY_TYPE,     &ulKeyType_Gost28147, sizeof(CK_ULONG)},
    {CKA_TOKEN,        &blFalse,         sizeof(CK_BBOOL)},
    {CKA_SENSITIVE,    &blFalse,         sizeof(CK_BBOOL)},
    {CKA_EXTRACTABLE, &blTrue,          sizeof(CK_BBOOL)},
    {CKA_ENCRYPT,       &blTrue,          sizeof(CK_BBOOL)},
    {CKA_DECRYPT,      &blTrue,          sizeof(CK_BBOOL)},
    {CKA_GOST28147PARAMS, gost28147params_A, sizeof(
gost28147params_A)},
};

CK_ULONG
    ulPubKeyCount =
    sizeof(caGOST_PublicKeyTemplate)/sizeof(CK_ATTRIBUTE),
    ulPriKeyCount =
    sizeof(caGOST_PrivateKeyTemplate)/sizeof(CK_ATTRIBUTE),
    ulDeriveKeyCount =
    sizeof(caDeriveKey)/sizeof(CK_ATTRIBUTE);

CK_OBJECT_HANDLE
    hSendPubKey = 0, // handle to public key of
sender
    hSendPriKey = 0, // handle to private key of
sender
    hRecpPubKey = 0, // handle to public key of
recipient
    hRecpPriKey = 0, // handle to private key of
recipient
    hSendDH_Key = 0, // Diffy–Hellman key of the
sender
    hRecpDH_Key = 0; // Diffy–Hellman key of the
recipient
CK_MECHANISM
    cmKeyGenMechanism, // mechanism for key pair
generation

```

```

        cmDeriveMechanism; // mechanism for key
    derivation

    CK_ULONG
        i;

    CK_BYTE_PTR
        pbSecretKeyParam = NULL,
        pbSend_PubKeyValue = NULL,
        pbRecp_PubKeyValue = NULL,
        pbSend_PriKeyValue = NULL,
        pbRecp_PriKeyValue = NULL,
        pbSendDH_Key_Value = NULL,
        pbRecpDH_Key_Value = NULL;

    CK_ATTRIBUTE
        caSecretKeyParam =
            {CKA_GOST28147PARAMS, pbSecretKeyParam, 0},
        caSend_PubKeyValue = {CKA_VALUE,
        pbSend_PubKeyValue, 0},
        caSend_PriKeyValue = {CKA_VALUE,
        pbSend_PriKeyValue, 0},
        caRecp_PubKeyValue = {CKA_VALUE,
        pbRecp_PubKeyValue, 0},
        caRecp_PriKeyValue = {CKA_VALUE,
        pbRecp_PriKeyValue, 0},
        SendDH_Key_Value = {CKA_VALUE, pbSendDH_Key_Value, 0},
        RecpDH_Key_Value = {CKA_VALUE, pbRecpDH_Key_Value, 0};

    // parameters for derivation mechanism
    CK_GOSTR3410_DERIVE_PARAMS_PTR DeriveParams;
    // UKM must be non-zero by RFC 4357
    CK_ULONG ulUKMLen = 16;
    CK_BYTE fixed_ukm[16] = {
        0x9d, 0x23, 0x98, 0xc0, 0x12, 0x31, 0x2a, 0x8e,
        0x23, 0xf4, 0xca, 0xdd, 0x03, 0xa8, 0x17, 0x97,
    };
    char *TextBlock =
    "This text block will be encrypted and the cipher text will be
    decrypted.";
    CK_ULONG ulDataSize = 0; // size of text data

    CK_BYTE_PTR pbCipherText = NULL; // encrypted data

```

```

CK_ULONG      ulCipherSize = 0;          // size of encrypted
data
CK_BYTE_PTR   pbDecryptedText = NULL;    // decrypted data
CK_ULONG      ulDecryptedDataSize = 0;   // size of decrypted
data
CK_CHAR *api_path = PKCS11_API_PATH;
CK_UTF8CHAR *user_pin = "01234567";
CK_ULONG slot_num = 0;
CK_CHAR *api_path2 = PKCS11_API_PATH;
CK_UTF8CHAR *user_pin2 = "01234567";
CK_ULONG slot_num2 = 0;
SYSTEMTIME    t1, t2;
CK_ULONG      diff/*, min_time, max_time, avg_time*/;

printf("Starting CKM_GOSTR3410_12_DERIVE (2012_512) test\n");

for (i=1; i<(CK_ULONG) argc; i++) {
    if (strcmp("-api", argv[i]) == 0) {
        ++i;
        api_path = argv[i];
    } else if (strcmp("-slot", argv[i]) == 0) {
        ++i;
        slot_num = atoi(argv[i]);
    } else if (strcmp("-user_pin", argv[i]) == 0) {
        ++i;
        user_pin = argv[i];
    }
}

api_path2 = api_path;
user_pin2 = user_pin;
slot_num2 = slot_num;

#ifdef WIN32
    hPkcsLib = LoadLibrary(api_path);
#else
    hPkcsLib = dlopen(api_path, RTLD_NOW);
#endif
    if ( hPkcsLib == NULL ) {
        printf(
            "Can't load PKCS#11 API library. "
            "Check API library path.\n");
#ifdef WIN32
        printf("dlerror: %s\n", dlerror());

```

```
#endif
    return -1;
}
#ifdef WIN32
    pcGetFunctionList =
        (CK_C_GetFunctionList)GetProcAddress(
            hPkesLib, "C_GetFunctionList");
#else
    pcGetFunctionList =
        (CK_C_GetFunctionList)dlsym(
            hPkesLib, "C_GetFunctionList");
#endif

// get PKCS #11 function list
rvResult = pcGetFunctionList(&Pkes11FuncList);
printf("Load PKCS #11 function list result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// initialize Cryptoki
rvResult = Pkes11FuncList->C_Initialize(NULL);
printf("Initialize Cryptoki result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// get slot list
rvResult = Pkes11FuncList->C_GetSlotList(CK_FALSE, NULL, &
    ulSlotCount);
printf("Get slot list result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

if (ulSlotCount > 0)
{
    // allocate memory for slot list
    pSlotList = (CK_SLOT_ID_PTR) malloc(
        ulSlotCount * sizeof(CK_SLOT_ID));

    rvResult =
        Pkes11FuncList->C_GetSlotList(
            CK_FALSE, pSlotList, &ulSlotCount);

    if (rvResult != CKR_OK) return rvResult;
    printf("Slot count: %d\n", ulSlotCount);
}
else return -3;
```

```
// get information about sender and recipient slots.
for(i=0; i<ulSlotCount; ++i)
{
    rvResult = Pkcs11FuncList->C_GetSlotInfo(pSlotList[i], &
SlotInfo);
    if (rvResult == CKR_OK)
    { // if a token is present in this slot
        if ((SlotInfo.flags & CKF_TOKEN_PRESENT) ==
CKF_TOKEN_PRESENT)
        {
            SlotId = pSlotList[i];
            break;
        }
    }
}
if (i>=ulSlotCount) {
    printf("No slots with token present\n");
    return -3;
}

printf("Slot ID: %d\n", SlotId);

// open session for slot with ID = SlotId[0]
rvResult = Pkcs11FuncList->C_OpenSession(SlotId,
(CKF_SERIAL_SESSION | CKF_RW_SESSION),
NULL,
0,
&hSession);
printf("Open session result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// user login
rvResult = Pkcs11FuncList->C_Login(hSession,
CKU_USER,
user_pin,
strlen(user_pin));
printf("Login result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;
// load library PKCS #11
#ifdef WIN32
    hPkcsLib2 = LoadLibrary(api_path2);
#else
    hPkcsLib2 = dlopen(api_path2, RTLD_NOW);
#endif
```



```
if ( hPkcsLib2 == NULL ) {
    printf("Can't load PKCS#11 API library. "
        "Check API library path.\n");
#ifdef WIN32
    printf("dlerror: %s\n", dlerror());
#endif
    return FALSE;
}
#ifdef WIN32
pcGetFunctionList2 =
    (CK_C_GetFunctionList)GetProcAddress(
        hPkcsLib2, "C_GetFunctionList");
#else
pcGetFunctionList2 =
    (CK_C_GetFunctionList)dlsym(
        hPkcsLib2, "C_GetFunctionList");
#endif

// get PKCS #11 function list
rvResult = pcGetFunctionList2(&Pkcs11FuncList2);
printf("Load PKCS #11 function list result: 0x%x\n", rvResult)
;
if (rvResult != CKR_OK) return rvResult;

// initialize Cryptoki
rvResult = Pkcs11FuncList2->C_Initialize(NULL);
printf("Initialize Cryptoki result: 0x%x\n", rvResult);
if (rvResult != CKR_OK &&
    rvResult != CKR_CRYPTOKI_ALREADY_INITIALIZED) return
rvResult;

// get slot list
rvResult = Pkcs11FuncList2->C_GetSlotList(CK_FALSE, NULL, &
    ulSlotCount2);
printf("Get slot list result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

if (ulSlotCount2 > 0)
{ // allocate memory for slot list
    pSlotList2 =
        (CK_SLOT_ID_PTR) malloc(
            ulSlotCount2 * sizeof(CK_SLOT_ID));

    rvResult =
```

```
Pkcs11FuncList2->C_GetSlotList(CK_FALSE,
    pSlotList2, &ulSlotCount2);

    if (rvResult != CKR_OK) return rvResult;
    printf("Slot count: %d\n", ulSlotCount2);
}
else return -3;

// get information about sender and recipient slots.
for(i=0; i<ulSlotCount2; ++i)
{
    rvResult = Pkcs11FuncList2->C_GetSlotInfo(
        pSlotList2[i], &SlotInfo);
    if (rvResult == CKR_OK)
    { // if a token is present in this slot
        if ((SlotInfo.flags & CKF_TOKEN_PRESENT) ==
            CKF_TOKEN_PRESENT)
        {
            SlotId2 = pSlotList2[i];
            break;
        }
    }
}
if (i >= ulSlotCount2) {
    printf("No slots with token present\n");
    return -3;
}

printf("Slot ID 2: %d\n", SlotId2);
{
    CK_TOKEN_INFO tinfo;
    rvResult = Pkcs11FuncList2->C_GetTokenInfo(pSlotList2[i], &
tinfo);
    if (rvResult != CKR_OK) {
        printf("Can't get token info\n");
        return -4;
    }
}

// open session for slot with ID = SlotId2[0]
rvResult = Pkcs11FuncList2->C_OpenSession(SlotId2,
    (CKF_SERIAL_SESSION | CKF_RW_SESSION),
    NULL,
    0,
```

```
&hSession2);
printf("Open session result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// User login may return CKR_USER_ALREADY_LOGGED_IN
// if the same token used for recipient.
rvResult = Pkcs11FuncList2->C_Login(hSession2,
    CKU_USER,
    user_pin2,
    strlen(user_pin2));
printf("Login result: 0x%x\n", rvResult);
if (rvResult != CKR_OK &&
    rvResult != CKR_USER_ALREADY_LOGGED_IN) return rvResult;

// generate sender key pair with GOST R 34.10-2012 (512)
printf("Generate key pair of sender\n");
printf("    mechanism type: CKM_GOSTR3410_512_KEY_PAIR_GEN\n");

cmKeyGenMechanism.mechanism = CKM_GOSTR3410_512_KEY_PAIR_GEN;
cmKeyGenMechanism.pParameter = NULL;
cmKeyGenMechanism.ulParameterLen = 0;
rvResult = Pkcs11FuncList->C_GenerateKeyPair(hSession,
    &cmKeyGenMechanism,
    caGOST_PublicKeyTemplate,
    ulPubKeyCount,
    caGOST_PrivateKeyTemplate,
    ulPriKeyCount,
    &hSendPubKey,
    &hSendPriKey);
printf("    generate sender key pair: result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    sender public key handle: %d\n",
    (unsigned long)hSendPubKey);
printf("    sender private key handle: %d\n",
    (unsigned long)hSendPriKey);

// generate recipient key pair with GOST R 34.10-2012 (512)
printf("Generate key pair of recipient\n");
printf("    mechanism type: CKM_GR3410_512_KEY_PAIR_GEN\n");

rvResult = Pkcs11FuncList2->C_GenerateKeyPair(hSession2,
    &cmKeyGenMechanism,
```

```
caGOST_PublicKeyTemplate ,
ulPubKeyCount ,
caGOST_PrivateKeyTemplate ,
ulPriKeyCount ,
&hRecpPubKey ,
&hRecpPriKey);
printf("    generate recipient key pair: result: 0x%x\n",
rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    recipient public key handle: %d\n",
(unsigned long)hRecpPubKey);
printf("    recipient private key handle: %d\n",
(unsigned long)hRecpPriKey);

// get value of sender public key
printf("Get value of sender public key\n");

rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession ,
hSendPubKey ,
&caSend_PubKeyValue ,
1);
if (rvResult == CKR_OK)
{
caSend_PubKeyValue.pValue =
(CK_BYTE_PTR) malloc(caSend_PubKeyValue.ulValueLen);
rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession ,
hSendPubKey ,
&caSend_PubKeyValue ,
1);
}
printf("    Get sender public key value: result: 0x%x\n",
rvResult);
if (rvResult != CKR_OK) return rvResult;

pbSend_PubKeyValue = (CK_BYTE_PTR) caSend_PubKeyValue.pValue;
printf("    Sender public key value: ");

for(i=0; i<caSend_PubKeyValue.ulValueLen; ++i)
printf("%x ", pbSend_PubKeyValue[i]);
printf("\n");

// get value of recipient public key
printf("Get value of recipient public key\n");
```

```

rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
    hRecpPubKey,
    &caRecp_PubKeyValue,
    1);
if (rvResult == CKR_OK)
{
    caRecp_PubKeyValue.pValue =
        (CK_BYTE_PTR) malloc(caRecp_PubKeyValue.ulValueLen);
    rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
        hRecpPubKey,
        &caRecp_PubKeyValue,
        1);
}

printf("    Get recipient public key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    Recipient public key value: ");
pbRecp_PubKeyValue = (CK_BYTE_PTR) caRecp_PubKeyValue.pValue;

for(i=0; i<caRecp_PubKeyValue.ulValueLen; ++i)
    printf("%x ", pbRecp_PubKeyValue[i]);
printf("\n");

// fill parameters for derivation mechanism
DeriveParams = (CK_GOSTR3410_DERIVE_PARAMS_PTR)
    malloc(sizeof(CK_GOSTR3410_DERIVE_PARAMS));
DeriveParams->kdf = CKD_CPDIVERSIFY_KDF;
DeriveParams->pPublicData = (CK_BYTE_PTR) caRecp_PubKeyValue.
    pValue;
DeriveParams->ulPublicDataLen = caRecp_PubKeyValue.ulValueLen;
DeriveParams->pUKM = fixed_ukm;
DeriveParams->ulUKMLen = sizeof(fixed_ukm);

cmDeriveMechanism.mechanism = CKM_GOSTR3410_12_DERIVE;
cmDeriveMechanism.pParameter = DeriveParams;
cmDeriveMechanism.ulParameterLen = sizeof(
    CK_GOSTR3410_DERIVE_PARAMS);

printf("Derive Diffie-Hellman key\n");
printf("    derivation mechanism: CKM_GOSTR3410_DERIVE\n");

// derive Diffie-Hellman key of sender

```

```

printf("    derive Diffie-Hellman key for sender\n");
    GetSystemTime(&t1);
rvResult = Pkcs11FuncList->C_DeriveKey(hSession,
    &cmDeriveMechanism,
    hSendPriKey,
    caDeriveKey,
    ulDeriveKeyCount,
    &hSendDH_Key);
    GetSystemTime(&t2);
    diff = process_time(t1, t2);
fprintf(stderr, "C_DeriveKey time: %ld msec\n", diff);
printf("    derive key result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    sender's Diffie-Hellman key handle: %d\n",
    (unsigned long)hSendDH_Key);

// get value of sender derived key
printf("Get value of sender derived key\n");
rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession,
    hSendDH_Key,
    &SendDH_Key_Value,
    1);
if (rvResult == CKR_OK)
{
    SendDH_Key_Value.pValue =
        (CK_BYTE_PTR) malloc(SendDH_Key_Value.ulValueLen);
    rvResult = Pkcs11FuncList->C_GetAttributeValue(hSession,
        hSendDH_Key,
        &SendDH_Key_Value,
        1);
}

printf("    Get sender derived key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

pbSendDH_Key_Value = (CK_BYTE_PTR) SendDH_Key_Value.pValue;

printf("    Sender derived key value: ");

for (i=0; i<SendDH_Key_Value.ulValueLen; ++i)
    printf("%x ", pbSendDH_Key_Value[i]);
printf("\n");

```

```
// derive Diffie-Hellman key of recipient
printf("    derive Diffie-Hellman key for recipient\n");
DeriveParams->pPublicData = (CK_BYTE_PTR) caSend_PubKeyValue.
    pValue;
DeriveParams->ulPublicDataLen = caSend_PubKeyValue.ulValueLen;

rvResult = Pkcs11FuncList2->C_DeriveKey(hSession2,
    &cmDeriveMechanism,
    hRecpPriKey,
    caDeriveKey,
    ulDeriveKeyCount,
    &hRecpDH_Key);
printf("    derive key result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    recipient's Diffie-Hellman key handle: %d\n",
    (unsigned long)hRecpDH_Key);

// get value of recipient derived key
printf("Get value of recipient derived key\n");
rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
    hRecpDH_Key,
    &RecpDH_Key_Value,
    1);
if (rvResult == CKR_OK)
{
    RecpDH_Key_Value.pValue =
        (CK_BYTE_PTR) malloc(RecpDH_Key_Value.ulValueLen);
    rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSession2,
        hRecpDH_Key,
        &RecpDH_Key_Value,
        1);
}

printf("    Get recipient derived key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

pbRecpDH_Key_Value = (CK_BYTE_PTR) RecpDH_Key_Value.pValue;

printf("    Recipient derived key value: ");

for (i=0; i<RecpDH_Key_Value.ulValueLen; ++i)
```

```
    printf("%x ", pbRecpDH_Key_Value[i]);
    printf("\n");

    // Sender and recipient DH key values must be equal
    if (RecpDH_Key_Value.ulValueLen != SendDH_Key_Value.ulValueLen
        ) {
        printf("Sender and recipient DH keys are of different length
        \n");
        return -1;
    }
    for(i=0; i<RecpDH_Key_Value.ulValueLen; ++i) {
        if (pbRecpDH_Key_Value[i] != pbSendDH_Key_Value[i]) {
            printf(
                "Sender and recipient KEK keys "
                "are different at position %d\n", i);
            return -1;
        }
    }
    printf("Sender and recipient KEK keys are equal\n");
    printf("CKM_GOSTR3410_12_DERIVE test SUCCESS\n");
    rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
        hSendPubKey);
    printf("    C_DestroyObject result: 0x%x\n", rvResult);
    rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
        hSendPriKey);
    printf("    C_DestroyObject result: 0x%x\n", rvResult);

    rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
        hRecpPubKey);
    printf("    C_DestroyObject result: 0x%x\n", rvResult);
    rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
        hRecpPriKey);
    printf("    C_DestroyObject result: 0x%x\n", rvResult);

    rvResult = Pkcs11FuncList->C_DestroyObject(hSession,
        hSendDH_Key);
    printf("    C_DestroyObject result: 0x%x\n", rvResult);

    rvResult = Pkcs11FuncList2->C_DestroyObject(hSession2,
        hRecpDH_Key);
    printf("    C_DestroyObject result: 0x%x\n", rvResult);
    rvResult = Pkcs11FuncList->C_Logout(hSession);
    printf("Sender logout result: 0x%x\n", rvResult);
```



```

// User logout may return CKR_USER_NOT_LOGGED_IN
// if the same token used for recipient.
rvResult = Pkcs11FuncList2->C_Logout(hSession2);
printf("Recipient logout result: 0x%x\n", rvResult);
// Close session
rvResult = Pkcs11FuncList->C_CloseSession(hSession);
printf("Sender close session result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList2->C_CloseSession(hSession2);
printf("Recipient close session result: 0x%x\n", rvResult);

printf("CKM_GOSTR3410_12_DERIVE (2012_256) test SUCCESS\n");
return 0;
}

```

3.3.8 Шифрование по ГОСТ 28147-89

В данной программе демонстрируется симметричное шифрование с помощью механизма CKM_GOST28147 с различными предопределенными в RFC 4357 наборами параметров. В одном из примеров используется также новый набор параметров "Z" (1.2.643.7.1.2.5.1.1), рекомендованный ТК 26 в качестве умалчиваемого [10]. Заметим, что для тестового набора параметров шифрования используется режим гаммирования CNT, а для остальных – режим гаммирования с обратной связью OFB. Кроме того, в данном примере демонстрируется сохранение/восстановление промежуточного состояния контекста шифрования в двух сессиях.

Размер состояния контекста шифрования зависит от реализации и в разных версиях библиотеки может оказаться различным, поэтому не следует надеяться на фиксированный размер буфера при сохранении этого состояния с помощью функции C_GetOperationState.

Листинг 3.17: ckm_gost28147.c

```

#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost28147(CK_SESSION_HANDLE sess, CK_SESSION_HANDLE
    sess2);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
    }
}

```

```

    return rc;
}
rc = test_crypto();
if (rc != CKR_OK) {
    fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
    return rc;
}
return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession, hSession2;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession2);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession 2 success\n");

    rc = funcs->C_GetMechanismInfo(SlotId, CKM_GOST28147, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n===== Mechanism CKM_GOST28147 not supported =====\n"
        );
    }
    else {
        fprintf(stderr,
            "\n===== CKM_GOST28147 test =====\n"
        );
    }
}

```

```

rc = test_gost28147(hSession, hSession2);
if (rc != CKR_OK) {
    fprintf(stderr,
        "ERROR CKM_GOST28147 failed, rc = 0x%x\n", rc);
} else {
    fprintf(stderr, "CKM_GOST28147 test passed.\n");
}
}

if( (rc = funcs->C_CloseSession(hSession2)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession 2 failed with 0x%x\n", rc);
}
else {
    fprintf(stderr, "C_CloseSession 2 success\n");
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr, "C_CloseSession success\n");
}

out:
    return rc;
}

CK_RV test_gost28147(CK_SESSION_HANDLE sess, CK_SESSION_HANDLE
    sess2)
{
    CK_RV rc = CKR_OK;
    CK_ULONG len, state_len;
    CK_BYTE value[1024], value2[1024];
    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    CK_BYTE *state;
    CK_MECHANISM mechanism_desc = {CKM_GOST28147, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;

```

```
static CK_OBJECT_CLASS param_oclass = CKO_DOMAIN_PARAMETERS;
static CK_KEY_TYPE key_type = CKK_GOST28147;

static CK_BYTE key [] = {
    0xc3, 0x73, 0x0c, 0x5c, 0xbc, 0xca, 0xcf, 0x91,
    0x5a, 0xc2, 0x92, 0x67, 0x6f, 0x21, 0xe8, 0xbd,
    0x4e, 0xf7, 0x53, 0x31, 0xd9, 0x40, 0x5e, 0x5f,
    0x1a, 0x61, 0xdc, 0x31, 0x30, 0xa6, 0x50, 0x11,
};
static CK_BYTE data [] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
};
static CK_BYTE data2 [200];

//par_cipher_0
static CK_BYTE oid1 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x00,
};
static CK_BYTE et1 [] = {
    0x08, 0x79, 0x76, 0x86, 0xee, 0xbb, 0x99, 0xe7,
    0x7f, 0xa3, 0x75, 0xff, 0x6b, 0x7c, 0x27, 0x6f,
};

//par_cipher_A
static CK_BYTE oid2 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
};
static CK_BYTE et2 [] = {
    0x49, 0x11, 0x96, 0x11, 0xad, 0x19, 0x12, 0x52,
    0x7c, 0x49, 0x5f, 0x23, 0xb9, 0x23, 0x62, 0xd9,
};

//par_cipher_B
static CK_BYTE oid3 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02,
};
static CK_BYTE et3 [] = {
    0xab, 0x45, 0x07, 0xb4, 0x96, 0xc5, 0x5b, 0xfd,
    0x61, 0x16, 0x64, 0x9d, 0x43, 0x87, 0x42, 0x53,
};

//par_cipher_C
static CK_BYTE oid4 [] = {
```

```

    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x03,
};
static CK_BYTE et4 [] = {
    0x98, 0x7c, 0x0c, 0x18, 0x3c, 0x64, 0xd5, 0x4b,
    0x75, 0x8e, 0xaf, 0x67, 0x52, 0x87, 0x15, 0x7b,
};

//par_cipher_D
static CK_BYTE oid5 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x04,
};
static CK_BYTE et5 [] = {
    0xa2, 0x4c, 0x7c, 0x4e, 0x91, 0xed, 0x19, 0x8c,
    0xdc, 0xf5, 0x1f, 0x07, 0x5c, 0x80, 0xf6, 0xa4,
};
//par_cipher_Z
static CK_BYTE oid6 [] = {
    0x06, 0x08, 0x2a, 0x85, 0x07, 0x01, 0x02, 0x05, 0x01, 0x01,
};
static CK_BYTE et6 [] = {
    0xea, 0x2f, 0x3f, 0x5f, 0xdb, 0x6f, 0x19, 0x01,
    0xba, 0x3d, 0x01, 0xcd, 0x2c, 0x64, 0x6d, 0xcb,
};

static CK_ATTRIBUTE key_template [] = {
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_GOST28147PARAMS, oid1, sizeof(oid1) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_DECRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VALUE, key, sizeof(key) }
};
static CK_BYTE ivec [8] = {1};

printf("Zero Data\n");
printf("Param Set 0\n");
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}
}

```

```
mechanism->pParameter = ivec;
mechanism->ulParameterLen = sizeof(ivec);
rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et1);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

printf("Param Set A\n");
key_template[2].pValue = oid2;
key_template[2].ulValueLen = sizeof(oid2);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
```

```
    goto end;
}

CHECK(value, len, et2);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

printf("Param Set B\n");
key_template[2].pValue = oid3;
key_template[2].ulValueLen = sizeof(oid3);
rc = funcs->C_CreateObject(sess,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et3);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

printf("Param Set C\n");
```

```
key_template[2].pValue = oid4;
key_template[2].ulValueLen = sizeof(oid4);
rc = funcs->C_CreateObject(sess,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et4);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

printf("Param Set D\n");
key_template[2].pValue = oid5;
key_template[2].ulValueLen = sizeof(oid5);
rc = funcs->C_CreateObject(sess,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
```



```
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et5);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

printf("Param Set Z\n");
key_template[2].pValue = oid6;
key_template[2].ulValueLen = sizeof(oid6);
rc = funcs->C_CreateObject(sess,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et6);
```

```
rc = funcs->C_DestroyObject(sess , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    goto end;
}

printf("Two Sessions With Get/SetOperationState\n");

printf("Random Data\n");
rc = funcs->C_GenerateRandom( sess , data2 , sizeof(data2) );
if (rc != CKR_OK) {
    fprintf(stderr , "C_GenerateRandom failed: 0x%x\n" , rc);
    goto end;
}

printf("Param Set B\n");
key_template[2].pValue = oid3;
key_template[2].ulValueLen = sizeof(oid3);
rc = funcs->C_CreateObject(sess ,
    key_template ,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE) , &keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_CreateObject failed: 0x%x\n" , rc);
    goto end;
}

rc = funcs->C_EncryptInit(sess , mechanism , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_EncryptInit failed: 0x%x\n" , rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_EncryptUpdate(sess , data2 , 192 , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr , "C_EncryptUpdate failed: 0x%x\n" , rc);
    goto end;
}

// Save session operation state
state_len = 0;
rc = funcs->C_GetOperationState(sess , NULL , &state_len);
state = malloc(state_len);
```

```
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
rc = funcs->C_GetOperationState(sess, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}

// Restore saved operation state to other session
rc = funcs->C_SetOperationState(sess2, state, state_len,
    keyh, CK_INVALID_HANDLE);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SetOperationState failed: 0x%x\n", rc);
    goto end;
}
memcpy(value2, value, sizeof(value));

free(state);
state_len = 0;
rc = funcs->C_GetOperationState(sess2, NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(sess2, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}

free(state);
state_len = 0;
rc = funcs->C_GetOperationState(sess, NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(sess, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
```

```
    goto end;
}
rc = funcs->C_SetOperationState(sess, state, state_len,
    keyh, CK_INVALID_HANDLE);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SetOperationState failed: 0x%x\n", rc);
    goto end;
}

    // Continue operations in first session
len = sizeof(value) - 192;
rc = funcs->C_EncryptUpdate(sess,
    data2 + 192, sizeof(data2) - 192, value + 192, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptUpdate failed: 0x%x\n", rc);
    goto end;
}

    // Continue restored operations in second session
len = sizeof(value2) - 192;
rc = funcs->C_EncryptUpdate(sess2,
    data2 + 192, sizeof(data2) - 192, value2 + 192, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptUpdate failed: 0x%x\n", rc);
    goto end;
}

    // Finalize operations in first session
len = sizeof(value) - sizeof(data2);
rc = funcs->C_EncryptFinal(sess, value + sizeof(data2), &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptFinal failed: 0x%x\n", rc);
    goto end;
}
    memcpy(value2, value, sizeof(value));

free(state);
state_len = 0;
rc = funcs->C_GetOperationState(sess2, NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
state = malloc(state_len);
```

```
rc = funcs->C_GetOperationState(sess2, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}

// Finalize operations in second session
len = sizeof(value2) - sizeof(data2);
rc = funcs->C_EncryptFinal(sess2, value2 + sizeof(data2), &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptFinal failed: 0x%x\n", rc);
    goto end;
}
// len == 0

rc = funcs->C_DecryptInit(sess2, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(data2);
rc = funcs->C_Decrypt(sess2, value2, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, data2);

rc = funcs->C_DecryptInit(sess2, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    goto end;
}

len = 192;
rc = funcs->C_DecryptUpdate(sess2, value2, 192, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptUpdate failed: 0x%x\n", rc);
    goto end;
}
len = sizeof(value2) - 192;
```

```
rc = funcs->C_DecryptUpdate(sess2 ,
    value2+192, sizeof(value2)-192, value+192, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptUpdate failed: 0x%x\n", rc);
    goto end;
}
len = sizeof(value2);
rc = funcs->C_DecryptFinal(sess2, value+sizeof(value2), &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptFinal failed: 0x%x\n", rc);
    goto end;
}

// len == 0
len = sizeof(data2);
CHECK(value, len, data2);

rc = funcs->C_DecryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    goto end;
}
free(state);
state_len = 0;
rc = funcs->C_GetOperationState(sess, NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(sess, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
rc = funcs->C_DecryptUpdate(sess, value2, 192, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptUpdate failed: 0x%x\n", rc);
    goto end;
}
free(state);
state_len = 0;
rc = funcs->C_GetOperationState(sess, NULL, &state_len);
if (rc != CKR_OK) {
```

```
fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
goto end;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(sess, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
rc = funcs->C_DecryptUpdate(sess,
    value2+192, sizeof(value2)-192, value+192, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptUpdate failed: 0x%x\n", rc);
    goto end;
}
free(state);
state_len = 0;
rc = funcs->C_GetOperationState(sess, NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(sess, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
rc = funcs->C_DecryptFinal(sess, value+sizeof(value2), &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptFinal failed: 0x%x\n", rc);
    goto end;
}
free(state);

// len == 0
len = sizeof(data2);
CHECK(value, len, data2);

printf("SUCCESS\n");
rc = CKR_OK;
end:
if (keyh != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess, keyh);
}
```

```
}  
  
    return rc;  
}
```

3.3.9 Шифрование в режиме ECB

Для шифрования в режиме простой замены ECB в стандарте определен специальный механизм CKM_GOST28147_ECB. В примере `ckm_gost28147_ecb` демонстрируется его использование. Заметим, что согласно RFC 4357[8], в объектах параметров домена могут быть заданы только режимы шифрования CNT(0), CFB(1) или CBC(2).

Листинг 3.18: `ckm_gost28147_ecb.c`

```
#include "test_common.h"  
  
CK_RV test_crypto();  
CK_RV test_gost28147_ecb(CK_SESSION_HANDLE sess);  
  
int main(int argc, char *argv[]) {  
    CK_RV rc = CKR_OK;  
  
    rc = test_main(argc, argv);  
    if (rc != CKR_OK) {  
        fprintf(stderr, "test_main failed: 0x%x\n", rc);  
        return rc;  
    }  
    rc = test_crypto();  
    if (rc != CKR_OK) {  
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);  
        return rc;  
    }  
    return CKR_OK;  
}  
  
CK_RV test_crypto()  
{  
    CK_RV rc, ret;  
    CK_SESSION_HANDLE hSession;  
    CK_MECHANISM_INFO minfo;  
  
    rc = funcs->C_OpenSession(SlotId,  
        CKF_RW_SESSION | CKF_SERIAL_SESSION,  
        NULL_PTR, NULL_PTR, &hSession);
```



```
if (rc != CKR_OK) {
    fprintf(stderr,
        "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
    goto out;
}
fprintf(stderr, "C_OpenSession success\n");
/*
// log in as normal user
rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
    user_pin));
if (rc != CKR_OK) {
    fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
        rc);
    goto out_close;
}
fprintf(stderr, "C_Login success\n");
*/
rc = funcs->C_GetMechanismInfo(SlotId, CKM_GOST28147_ECB, &
    minfo);
if (rc != CKR_OK) {
    fprintf(stderr,
        "\n==== Mechanism CKM_GOST28147_ECB not supported
        =====\n");
} else {
    fprintf(stderr,
        "\n==== CKM_GOST28147_ECB test
        =====\n");
    rc = test_gost28147_ecb(hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR CKM_GOST28147_ECB failed, rc = 0x%x\n", rc);
    } else {
        fprintf(stderr, "CKM_GOST28147_ECB test passed.\n");
    }
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr, "C_CloseSession success\n");
}
}
```

```

out:
    return rc;
}

CK_RV test_gost28147_ecb(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[256];
    CK_ULONG len;
    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc = {CKM_GOST28147_ECB, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_KEY_TYPE key_type = CKK_GOST28147;

    static CK_BYTE key[] = {
        0xc3, 0x73, 0x0c, 0x5c, 0xbc, 0xca, 0xcf, 0x91,
        0x5a, 0xc2, 0x92, 0x67, 0x6f, 0x21, 0xe8, 0xbd,
        0x4e, 0xf7, 0x53, 0x31, 0xd9, 0x40, 0x5e, 0x5f,
        0x1a, 0x61, 0xdc, 0x31, 0x30, 0xa6, 0x50, 0x11 };
    // Данные для шифрования в блочных режимах должны поступать на
    // вход
    // порциями, длина которых кратна размеру блока, т.е. 8-ми.
    static CK_BYTE data[] = {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    };

    static CK_BYTE oid1[] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x00 };
    static CK_BYTE et1[] = {
        0xf4, 0xe5, 0x5c, 0xf1, 0xf0, 0xb5, 0xd6, 0x29,
        0xf4, 0xe5, 0x5c, 0xf1, 0xf0, 0xb5, 0xd6, 0x29 };

    static CK_BYTE oid2[] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01 };
    static CK_BYTE et2[] = {
        0xd8, 0xea, 0xd4, 0xbd, 0xb6, 0xb5, 0x43, 0x26,
        0xd8, 0xea, 0xd4, 0xbd, 0xb6, 0xb5, 0x43, 0x26 };
}

```

```

static CK_BYTE oid3 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02 };
static CK_BYTE et3 [] = {
    0x7b, 0x18, 0xc6, 0x12, 0x5d, 0xfd, 0xe0, 0x80,
    0x7b, 0x18, 0xc6, 0x12, 0x5d, 0xfd, 0xe0, 0x80 };

static CK_BYTE oid4 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x03 };
static CK_BYTE et4 [] = {
    0x3d, 0xac, 0xa3, 0x01, 0x9c, 0xee, 0x92, 0x14,
    0x3d, 0xac, 0xa3, 0x01, 0x9c, 0xee, 0x92, 0x14 };

static CK_BYTE oid5 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x04 };
static CK_BYTE et5 [] = {
    0x6b, 0x63, 0x55, 0xec, 0x8c, 0x78, 0x62, 0x46,
    0x6b, 0x63, 0x55, 0xec, 0x8c, 0x78, 0x62, 0x46 };

static CK_ATTRIBUTE key_template [] = {
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_GOST28147PARAMS, oid1, sizeof(oid1) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_DECRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VALUE, key, sizeof(key) }
};
CK_BYTE dummy [8];
CK_ULONG dummy_len = 0;
CK_BYTE key_to_wrap [] = {
    0xc2, 0x8f, 0xf3, 0xe4, 0x3a, 0xe2, 0x72, 0xd2,
    0x41, 0x10, 0xff, 0x88, 0x3b, 0x5c, 0x16, 0x2b,
    0x51, 0xd9, 0x4d, 0x4d, 0xd5, 0x1f, 0x67, 0x0a,
    0x58, 0x95, 0x40, 0xf2, 0x54, 0x97, 0x4d, 0x61,
};

CK_BYTE wrapping_key [] = {
    0x28, 0xff, 0x2f, 0xbd, 0x65, 0x1e, 0x31, 0x25,
    0xef, 0xa6, 0x4c, 0xdd, 0x91, 0x78, 0xd1, 0x89,
    0x55, 0x96, 0xe0, 0x3c, 0x7f, 0x9d, 0xe5, 0xe9,
    0xa5, 0x9d, 0x5a, 0x9a, 0x2b, 0x1c, 0x6f, 0x6c,
};

CK_BYTE wrapped_key [] = {
    0xb6, 0xa6, 0xf7, 0x09, 0x52, 0x09, 0xea, 0xfd,

```

```

0x78, 0xa3, 0x4d, 0xbd, 0x0c, 0xbd, 0x35, 0xc2,
0x5f, 0xee, 0x6c, 0xf0, 0x17, 0xa2, 0xd8, 0x02,
0x5b, 0x36, 0xf5, 0xb3, 0xd8, 0x0a, 0xca, 0x87,
};

CK_ATTRIBUTE wrapping_key_template[] = {
    { CKA_VALUE, wrapping_key, sizeof(wrapping_key) },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_GOST28147PARAMS, oid2, sizeof(oid2) },
    { CKA_SIGN, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VERIFY, &ltrue, sizeof(CK_BBOOL) },
};

rc = funcs->C_CreateObject(sess,
    wrapping_key_template,
    sizeof(wrapping_key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    return rc;
}

// Данные для шифрования в блочных режимах должны поступать на
// вход
// порциями, длина которых кратна размеру блока, т.е. 8-ми.
len = sizeof(value);
rc = funcs->C_EncryptUpdate(sess,
    key_to_wrap, sizeof(key_to_wrap), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptUpdate failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptFinal(sess, dummy, &dummy_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptFinal failed: 0x%x\n", rc);
    return rc;
}

```

```
CHECK(value , len , wrapped_key);
rc = funcs->C_DestroyObject(sess , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}

rc = funcs->C_CreateObject(sess ,
    key_template , sizeof(key_template)/sizeof(CK_ATTRIBUTE) ,
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_CreateObject failed: 0x%x\n" , rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess , mechanism , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_EncryptInit failed: 0x%x\n" , rc);
    return rc;
}

// Данные для шифрования в блочных режимах должны поступать на
// вход
// порциями, длина которых кратна размеру блока, т.е. 8-ми.
len = sizeof(value);
rc = funcs->C_EncryptUpdate(sess , data , sizeof(data) , value , &
    len);
if (rc != CKR_OK) {
    fprintf(stderr , "C_EncryptUpdate failed: 0x%x\n" , rc);
    return rc;
}
rc = funcs->C_EncryptFinal(sess , dummy , &dummy_len);
if (rc != CKR_OK) {
    fprintf(stderr , "C_EncryptFinal failed: 0x%x\n" , rc);
    return rc;
}

CHECK(value , len , et1);

rc = funcs->C_DecryptInit(sess , mechanism , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DecryptInit failed: 0x%x\n" , rc);
    return rc;
}
```

```
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, data);
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

key_template[2].pValue = oid2;
key_template[2].ulValueLen = sizeof(oid2);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, et2);

rc = funcs->C_DecryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    return rc;
}
```

```
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, data);
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

key_template[2].pValue = oid3;
key_template[2].ulValueLen = sizeof(oid3);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, et3);

rc = funcs->C_DecryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    return rc;
}
```

```
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, data);
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

key_template[2].pValue = oid4;
key_template[2].ulValueLen = sizeof(oid4);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, et4);

rc = funcs->C_DecryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    return rc;
}
```



```
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, data);
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

key_template[2].pValue = oid5;
key_template[2].ulValueLen = sizeof(oid5);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, et5);

rc = funcs->C_DecryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    return rc;
}
```

```
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, data);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

return rc;
}
```

3.3.10 Шифрование в режиме CBC

В данном примере демонстрируется не только собственно шифрование в режиме сцепления блоков CBC, но и организация этого шифрования с помощью объекта параметров домена. Заметим, что согласно RFC 4357[8], в объектах параметров домена могут быть заданы режимы шифрования CNT(0), CFB(1) или CBC(2).

Листинг 3.19: ckm_gost28147_cbc.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost28147_cbc(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
    }
}
```

```

    return rc;
}
return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
/*
    // log in as normal user
    rc = funcs->C_Login(
        hSession, CKU_USER, user_pin, strlen(user_pin));
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_Login failed, rc = 0x%x\n", rc);
        goto out_close;
    }
    fprintf(stderr, "C_Login success\n");
*/

    rc = funcs->C_GetMechanismInfo(SlotId, CKM_GOST28147, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n==== Mechanism CKM_GOST28147 not supported =====\n"
        );
    } else {
        fprintf(stderr,
            "\n==== CKM_GOST28147 CBC test
            =====\n");
        rc = test_gost28147_cbc(hSession);
        if (rc != CKR_OK) {
            fprintf(stderr,

```

```

        "ERROR CKM_GOST28147 CBC failed, rc = 0x%x\n", rc);
    } else {
        fprintf(stderr, "CKM_GOST28147 CBC test passed.\n");
    }
}

if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr, "C_CloseSession success\n");
}
}

out:
    return rc;
}

// В данном тесте проверяется не только собственно шифрование
// в режиме CBC, но и организация этого шифрования с помощью
// объекта параметров домена.
CK_RV test_gost28147_cbc(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[256];
    CK_ULONG len;
    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc = {CKM_GOST28147, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_OBJECT_CLASS dp_class = CKO_DOMAIN_PARAMETERS;
    static CK_KEY_TYPE key_type = CKK_GOST28147;

    static CK_BYTE 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,
    };
    static CK_BYTE iv [] = {

```

```

    0x90, 0x4B, 0x9C, 0x7A, 0xBB, 0x85, 0x0E, 0x87,
};
// Данные для шифрования в блочных режимах должны
// поступать на вход порциями, длина которых
// кратна размеру блока, т.е. 8-ми байтам.
// Паддинг в LCC не работает, поэтому его организация
// должна производиться на прикладном уровне.
static CK_BYTE data [] = {
    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,
// PKCS#5 padding bytes
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
};

static CK_BYTE et1 [] = {
    0xC2, 0xE2, 0x9F, 0x15, 0x98, 0x47, 0x87, 0x97,
    0x91, 0x85, 0x29, 0x8D, 0x28, 0xBB, 0x37, 0x54,
    0xBC, 0x8E, 0x77, 0xC3, 0x82, 0x72, 0x4D, 0x60,
    0x0B, 0x09, 0x0C, 0xEC, 0x2A, 0x26, 0x95, 0xE5,
    0xBF, 0x13, 0x01, 0xFA, 0x09, 0x51, 0xBE, 0x6F,
    0x81, 0x73, 0xAB, 0xF3, 0xCB, 0x11, 0x20, 0xB8,
    0x61, 0x5F, 0xA8, 0x5B, 0xF0, 0x75, 0xB3, 0x98,
    0x4B,
// encrypted PKCS#5 padding bytes
    0x62, 0x89, 0x18, 0xBC, 0x34, 0xDE, 0xC8,
};

static CK_ATTRIBUTE key_template [] = {
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_GOST28147PARAMS, NULL, 0 },
    { CKA_VALUE, key, sizeof(key) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_DECRYPT, &ltrue, sizeof(CK_BBOOL) },
};

const CK_OBJECT_CLASS oc_dp = CKO_DOMAIN_PARAMETERS;

CK_ATTRIBUTE t_find_dp [] = {

```

```

{CKA_CLASS,      (CK_VOID_PTR)&oc_dp,  sizeof(oc_dp)},
{CKA_KEY_TYPE,  &key_type,  sizeof(key_type)},
{CKA_OBJECT_ID, NULL, 0},
};

// Фактически это набор параметров А из RFC 4357,
// в котором режим шифрования изменен на CBC.
CK_BYTE dp_cbc [] =
{
    0x30, 0x53,
    0x04, 0x40,
    // +4 (таблица замен)
    0x93, 0xee, 0xb3, 0x1b, 0x67, 0x47, 0x5a, 0xda,
    0x3e, 0x6a, 0x1d, 0x2f, 0x29, 0x2c, 0x9c, 0x95,
    0x88, 0xbd, 0x81, 0x70, 0xba, 0x31, 0xd2, 0xac,
    0x1f, 0xd3, 0xf0, 0x6e, 0x70, 0x89, 0x0b, 0x08,
    0xa5, 0xc0, 0xe7, 0x86, 0x42, 0xf2, 0x45, 0xc2,
    0xe6, 0x5b, 0x29, 0x43, 0xfc, 0xa4, 0x34, 0x59,
    0xcb, 0x0f, 0xc8, 0xf1, 0x04, 0x78, 0x7f, 0x37,
    0xdd, 0x15, 0xae, 0xbd, 0x51, 0x96, 0x66, 0xe4,
    0x02, 0x01,
    // +70 (mode: 0-CNT, 1-CFB, 2-CBC)
    0x02,
    0x02, 0x01, 0x40,
    0x30, 0x09,
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x0e,
    // +84 (key meshing: 0-Null, 1-CryptoPro)
    0x01,
};
CK_ULONG uldp_cbc = sizeof(dp_cbc);
// OID и название не являются предопределенными в LCC
CK_BYTE oid_cbc [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x41
};
CK_CHAR *dp_label =
    "id-Gost28147-89-CryptoPro-A-ParamSet-CBC";

CK_ATTRIBUTE t_dp_cbc [] = {
    {CKA_CLASS, &dp_class, sizeof(dp_class)},
    {CKA_TOKEN, &lfalse, sizeof(lfalse)},
    {CKA_KEY_TYPE, &key_type, sizeof(key_type)},
    {CKA_OBJECT_ID, oid_cbc, sizeof(oid_cbc)},
    {CKA_VALUE, dp_cbc, uldp_cbc},
    {CKA_LABEL, dp_label, strlen(dp_label)+1},
};

```

```
};
CK_OBJECT_HANDLE hDP_CBC = CK_INVALID_HANDLE;

rc = funcs->C_CreateObject(sess, t_dp_cbc,
    sizeof(t_dp_cbc)/sizeof(CK_ATTRIBUTE),
    &hDP_CBC);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

// Если бы объект параметров домена не был создан,
// то следующая операция создания ключа для такого OID
// завершилась бы с ошибкой.
key_template[2].pValue = oid_cbc;
key_template[2].ulValueLen = sizeof(oid_cbc);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}
fprintf(stderr, "CBC domain parameters object created\n");

mechanism->pParameter = iv;
mechanism->ulParameterLen = sizeof(iv);
rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    goto end;
}

fprintf(stderr, "Plain text to encrypt:\n");
print_hex(data, sizeof(data));
// Данные для шифрования в блочных режимах должны
// поступать на вход порциями, длина которых
// кратна размеру блока, т.е. 8-ми.
len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    goto end;
}
}
```

```
if (len != sizeof(et1)) {
    fprintf(stderr, "Invalid ciphertext length: %d\n", len);
    rc = -1;
    goto end;
}
if (memcmp(value, et1, len) != 0) {
    fprintf(stderr, "Invalid ciphertext\n");
    rc = -2;
    goto end;
}
fprintf(stderr, "CBC encryption result OK\n");
print_hex(value, len);
rc = funcs->C_DecryptInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    goto end;
}
fprintf(stderr, "Decrypted text:\n");
print_hex(value, len);

if (len != sizeof(data)) {
    fprintf(stderr, "Invalid decrypted text length: %d\n", len);
    rc = -3;
    goto end;
}
if (memcmp(value, data, len) != 0) {
    fprintf(stderr, "Invalid decrypted text\n");
    rc = -4;
    goto end;
}
fprintf(stderr, "CBC decryption result OK\n");

end:
if (keyh != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess, keyh);
}
if (hDP_CBC != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess, hDP_CBC);
}
```



```
}  
  
    return rc;  
}
```

3.3.11 Шифрование в режиме CNT

Механизма CKM_GOST28147_CNT в стандарте нет. Режим CNT реализуется средствами стандарта весьма замысловатым образом через специальный объект параметров домена для механизма CKM_GOST28147. Заметим, что режим шифрования CNT необходим для поддержки русского шифр-сюта для протокола TLS. В проекте LS11SW нестандартный механизм CKM_GOST28147_CNT также добавлен для удобства использования. В примере `ckm_gost28147_cnt` демонстрируются различные способы шифрования в режиме CNT.

Листинг 3.20: `ckm_gost28147_cnt.c`

```
#include "test_common.h"  
  
CK_RV test_crypto();  
CK_RV test_gost28147_cnt(CK_SESSION_HANDLE sess);  
  
int main(int argc, char *argv[]) {  
    CK_RV rc = CKR_OK;  
  
    rc = test_main(argc, argv);  
    if (rc != CKR_OK) {  
        fprintf(stderr, "test_main failed: 0x%x\n", rc);  
        return rc;  
    }  
    rc = test_crypto();  
    if (rc != CKR_OK) {  
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);  
        return rc;  
    }  
    return CKR_OK;  
}  
  
CK_RV test_crypto()  
{  
    CK_RV rc, ret;  
    CK_SESSION_HANDLE hSession;  
    CK_MECHANISM_INFO minfo;
```

```

rc = funcs->C_OpenSession( SlotId ,
    CKF_RW_SESSION | CKF_SERIAL_SESSION,
    NULL_PTR, NULL_PTR, &hSession);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
    goto out;
}
fprintf(stderr , "C_OpenSession success\n");
/*
// log in as normal user
rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
    user_pin));
if (rc != CKR_OK) {
    fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
        rc);
    goto out_close;
}
fprintf(stderr, "C_Login success\n");
*/
rc = funcs->C_GetMechanismInfo( SlotId , CKM_GOST28147, &minfo);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "\n===== Mechanism CKM_GOST28147 not supported =====\n"
    );
} else {
    fprintf(stderr ,
        "\n===== CKM_GOST28147 CNT test
        =====\n");
    rc = test_gost28147_cnt(hSession);
    if (rc != CKR_OK) {
        fprintf(stderr ,
            "ERROR CKM_GOST28147 CNT failed, rc = 0x%x\n", rc);
    } else {
        fprintf(stderr , "CKM_GOST28147 CNT test passed.\n");
    }
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr ,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {

```

```

    fprintf(stderr, "C_CloseSession success\n");
}

out:
    return rc;
}

// Шифрование в режиме CNT может быть реализовано двумя способам
// и:
// - с помощью нестандартного механизма CKM_GOST28147_CNT;
// - с использованием нестандартных параметров домена (как в тес
// те CBC).
CK_RV test_gost28147_cnt(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_ULONG len, state_len, dummy_len;
    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    CK_BYTE *state;
    CK_BYTE iv[] = {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    };
    CK_MECHANISM mechanism_desc = {CKM_GOST28147, iv, sizeof(iv)};
};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;
    CK_MECHANISM mechanism_desc_cnt = {CKM_GOST28147_CNT, iv,
    sizeof(iv)};
    CK_MECHANISM_PTR mechanism_cnt = &mechanism_desc_cnt;

    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_KEY_TYPE key_type = CKK_GOST28147;

    static CK_BYTE key[] = {
        0xc3, 0x73, 0x0c, 0x5c, 0xbc, 0xca, 0xcf, 0x91,
        0x5a, 0xc2, 0x92, 0x67, 0x6f, 0x21, 0xe8, 0xbd,
        0x4e, 0xf7, 0x53, 0x31, 0xd9, 0x40, 0x5e, 0x5f,
        0x1a, 0x61, 0xdc, 0x31, 0x30, 0xa6, 0x50, 0x11 };
    static CK_BYTE data[] = {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    };
    CK_BYTE value[32];
    CK_BYTE buf[32];

```

```
static CK_BYTE oid1 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x00 };
static CK_BYTE oid_cnt1 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x40 };
static CK_BYTE et1 [] = {
    0x75, 0xea, 0xe3, 0x45, 0xb5, 0x12, 0xac, 0x47,
    0xe5, 0xe6, 0x33, 0x04, 0xe0, 0xb9, 0x99, 0xa6 };

static CK_BYTE oid2 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01 };
static CK_BYTE oid_cnt2 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x41 };
static CK_BYTE et2 [] = {
    0xd4, 0x7d, 0xab, 0xaa, 0x4b, 0x49, 0x3a, 0x8d,
    0xd6, 0xeb, 0xb6, 0x17, 0xd6, 0xa4, 0xfd, 0x31 };

static CK_BYTE oid3 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02 };
static CK_BYTE oid_cnt3 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x42 };
static CK_BYTE et3 [] = {
    0x22, 0xb3, 0x6e, 0x02, 0x7b, 0x03, 0xa5, 0x6a,
    0x5f, 0x23, 0xf4, 0xbd, 0x63, 0x6e, 0x03, 0x1f };

static CK_BYTE oid4 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x03 };
static CK_BYTE oid_cnt4 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x43 };
static CK_BYTE et4 [] = {
    0xf1, 0x26, 0xfc, 0x4d, 0x20, 0xd5, 0xe2, 0xcd,
    0xe8, 0x1c, 0x7e, 0x10, 0xed, 0x27, 0x07, 0xe0 };

static CK_BYTE oid5 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x04 };
static CK_BYTE oid_cnt5 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x44 };
static CK_BYTE et5 [] = {
    0xfd, 0x65, 0x30, 0xa2, 0xe2, 0x93, 0xfe, 0xc1,
    0xfe, 0x71, 0xc5, 0x79, 0x3d, 0x20, 0x66, 0x73 };

static CK_ATTRIBUTE key_template [] = {
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
```

```

    { CKA_GOST28147PARAMS, NULL, 0 },
    { CKA_VALUE, key, sizeof(key) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_DECRYPT, &ltrue, sizeof(CK_BBOOL) },
};
static CK_OBJECT_CLASS dp_class = CKO_DOMAIN_PARAMETERS;
const CK_OBJECT_CLASS oc_dp = CKO_DOMAIN_PARAMETERS;

CK_ATTRIBUTE t_find_dp[] = {
    {CKA_CLASS, (CK_VOID_PTR)&oc_dp, sizeof(oc_dp)},
    {CKA_KEY_TYPE, &key_type, sizeof(key_type)},
    {CKA_OBJECT_ID, NULL, 0},
};

// Фактически это набор параметров А из RFC 4357,
// в котором режим шифрования изменен на CNT.
CK_BYTE dp_cnt[] =
{
    0x30, 0x53,
    0x04, 0x40,
    // +4 (таблица замен)
    0x93, 0xee, 0xb3, 0x1b, 0x67, 0x47, 0x5a, 0xda,
    0x3e, 0x6a, 0x1d, 0x2f, 0x29, 0x2c, 0x9c, 0x95,
    0x88, 0xbd, 0x81, 0x70, 0xba, 0x31, 0xd2, 0xac,
    0x1f, 0xd3, 0xf0, 0x6e, 0x70, 0x89, 0x0b, 0x08,
    0xa5, 0xc0, 0xe7, 0x86, 0x42, 0xf2, 0x45, 0xc2,
    0xe6, 0x5b, 0x29, 0x43, 0xfc, 0xa4, 0x34, 0x59,
    0xcb, 0x0f, 0xc8, 0xf1, 0x04, 0x78, 0x7f, 0x37,
    0xdd, 0x15, 0xae, 0xbd, 0x51, 0x96, 0x66, 0xe4,
    0x02, 0x01,
    // +70 (mode: 0-CNT, 1-CFB, 2-CBC)
    0x00,
    0x02, 0x01, 0x40,
    0x30, 0x09,
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x0e,
    // +84 (key meshing: 0-Null, 1-CryptoPro)
    0x01,
};
CK_ULONG uldp_cnt = sizeof(dp_cnt);
// Этот OID и название не являются предопределенными в LCC,
// а используются для определения дополнительных параметров до
// мена.
CK_BYTE oid_cnt[] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x42};

```

```

CK_CHAR *dp_label = "id-Gost28147-89-CryptoPro-A-ParamSet-CNT"
;

CK_ATTRIBUTE t_dp_cnt [] = {
    {CKA_CLASS, &dp_class, sizeof(dp_class)},
    {CKA_TOKEN, &lfalse, sizeof(lfalse)},
    {CKA_KEY_TYPE, &key_type, sizeof(key_type)},
    {CKA_OBJECT_ID, oid_cnt, sizeof(oid_cnt)},
    {CKA_VALUE, dp_cnt, uldp_cnt},
    {CKA_LABEL, dp_label, strlen(dp_label)+1},
    {CKA_COPYABLE, &ltrue, sizeof(ltrue)}
};

CK_BYTE dp_value[4096];
CK_ULONG uldp_value = sizeof(dp_value);

CK_ATTRIBUTE t_dp_value = {
    CKA_VALUE, dp_value, sizeof(dp_value)
};

CK_ATTRIBUTE t_copy_dp_cnt [] = {
    {CKA_TOKEN, &lfalse, sizeof(lfalse)},
    {CKA_OBJECT_ID, NULL, 0},
    {CKA_VALUE, dp_value, 0},
};

CK_OBJECT_HANDLE hDP_CNT;
CK_ULONG ulCB = 0;

key_template[2].pValue = oid1;
key_template[2].ulValueLen = sizeof(oid1);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess, mechanism_cnt, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    return rc;
}

```

```
len = sizeof(value);
rc = funcs->C_EncryptUpdate(sess, data, 4, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptUpdate failed: 0x%x\n", rc);
    return rc;
}
// Save partial encryption result
memcpy(buf, value, 4);
state_len = 0;
rc = funcs->C_GetOperationState(sess, NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
state = malloc(state_len);
rc = funcs->C_GetOperationState(sess, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SetOperationState(sess, state, state_len,
    keyh, CK_INVALID_HANDLE);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SetOperationState failed: 0x%x\n", rc);
    goto end;
}
rc = funcs->C_EncryptUpdate(sess,
    data+4, sizeof(data)-4, value+4, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptUpdate failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_EncryptFinal(sess, value+sizeof(data), &
    dummy_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptFinal failed: 0x%x\n", rc);
    return rc;
}

len = sizeof(data);
CHECK(value, len, et1);

rc = funcs->C_DecryptInit(sess, mechanism_cnt, keyh);
```

```

if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_DecryptUpdate(sess, value, 7, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptUpdate failed: 0x%x\n", rc);
    return rc;
}
printf("len = %d, Decryption buffer:\n", len);
print_hex(value, sizeof(value));
rc = funcs->C_DecryptUpdate(sess,
    value+7, sizeof(data)-7, value+7, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptUpdate failed: 0x%x\n", rc);
    return rc;
}
printf("len = %d, Decryption buffer:\n", len);
print_hex(value, sizeof(value));
rc = funcs->C_DecryptFinal(sess,
    value+sizeof(data), &dummy_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptFinal failed: 0x%x\n", rc);
    return rc;
}
len = sizeof(data);
printf("len = %d, Decrypted data:\n", len);
print_hex(value, len);
printf("Decryption buffer:\n");
print_hex(value, sizeof(value));

CHECK(value, len, data);

// Restore partial encryption result
memcpy(value, buf, 4);
rc = funcs->C_SetOperationState(sess, state, state_len,
    keyh, CK_INVALID_HANDLE);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SetOperationState failed: 0x%x\n", rc);
    goto end;
}
rc = funcs->C_EncryptUpdate(sess,
    data+4, sizeof(data)-4, value+4, &len);
if (rc != CKR_OK) {

```



```
fprintf(stderr, "C_EncryptUpdate failed: 0x%x\n", rc);
return rc;
}
rc = funcs->C_EncryptFinal(sess,
value+sizeof(data), &dummy_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptFinal failed: 0x%x\n", rc);
    return rc;
}
len = sizeof(data);
CHECK(value, len, et1);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject error, code 0x%X\n", rc);
    return rc;
}
key_template[2].pValue = oid2;
key_template[2].ulValueLen = sizeof(oid2);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess, mechanism_cnt, keyh);
if (rc == CKR_FUNCTION_NOT_SUPPORTED) {
    fprintf(stderr,
"Non-standard parameter set OID not supported for CKM_GOST28147\
n");
} else {
    len = sizeof(value);
    rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len)
;
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
        return rc;
    }

    CHECK(value, len, et2);

    rc = funcs->C_DecryptInit(sess, mechanism_cnt, keyh);
```

```
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, data);
}
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject error, code 0x%X\n", rc);
    return rc;
}

// Теперь то же самое, но с использованием сессионного объекта
// параметров домена с нестандартным OID.
dp_cnt[70] = 0x00; // CNT mode

rc = funcs->C_CreateObject(sess,
    t_dp_cnt, sizeof(t_dp_cnt)/sizeof(CK_ATTRIBUTE),
    &hDP_CNT);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

key_template[2].pValue = oid_cnt;
key_template[2].ulValueLen = sizeof(oid_cnt);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_EncryptInit(sess, mechanism, keyh);
if (rc == CKR_FUNCTION_NOT_SUPPORTED) {
    fprintf(stderr,
```

```
"Non-standard parameter set OID not supported for CKM_GOST28147\
n");
} else {
    len = sizeof(value);
    rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len)
;
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
        return rc;
    }

    CHECK(value, len, et2);

    rc = funcs->C_DecryptInit(sess, mechanism, keyh);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
        return rc;
    }

    rc = funcs->C_Decrypt(sess, value, len, value, &len);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
        return rc;
    }

    CHECK(value, len, data);
}
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject error, code 0x%X\n", rc);
    return rc;
}

key_template[2].pValue = oid3;
key_template[2].ulValueLen = sizeof(oid3);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}
```

```
rc = funcs->C_EncryptInit(sess , mechanism_cnt , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_EncryptInit failed: 0x%x\n" , rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess , data , sizeof(data) , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr , "C_Encrypt failed: 0x%x\n" , rc);
    return rc;
}

CHECK(value , len , et3);

rc = funcs->C_DecryptInit(sess , mechanism_cnt , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DecryptInit failed: 0x%x\n" , rc);
    return rc;
}

rc = funcs->C_Decrypt(sess , value , len , value , &len);
if (rc != CKR_OK) {
    fprintf(stderr , "C_Decrypt failed: 0x%x\n" , rc);
    return rc;
}

CHECK(value , len , data);
rc = funcs->C_DestroyObject(sess , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject error, code 0x%X\n" , rc);
    return rc;
}

key_template[2].pValue = oid4;
key_template[2].ulValueLen = sizeof(oid4);
rc = funcs->C_CreateObject(sess ,
    key_template , sizeof(key_template)/sizeof(CK_ATTRIBUTE) , &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_CreateObject failed: 0x%x\n" , rc);
    return rc;
}
}
```

```
rc = funcs->C_EncryptInit(sess, mechanism_cnt, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, et4);

rc = funcs->C_DecryptInit(sess, mechanism_cnt, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, data);
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject error, code 0x%X\n", rc);
    return rc;
}

key_template[2].pValue = oid5;
key_template[2].ulValueLen = sizeof(oid5);
rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}
```

```
}

rc = funcs->C_EncryptInit(sess, mechanism_cnt, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_EncryptInit failed: 0x%x\n", rc);
    return rc;
}

len = sizeof(value);
rc = funcs->C_Encrypt(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Encrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, et5);

rc = funcs->C_DecryptInit(sess, mechanism_cnt, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DecryptInit failed: 0x%x\n", rc);
    return rc;
}

rc = funcs->C_Decrypt(sess, value, len, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Decrypt failed: 0x%x\n", rc);
    return rc;
}

CHECK(value, len, data);
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject error, code 0x%X\n", rc);
    return rc;
}

end:
return rc;
}
```

3.3.12 Генерация имитовставки

В данном примере имитовставка генерируется различными способами и с различными параметрами с помощью механизма `CKM_GOST28147_MAC`.

Генерация имитовставки - это контекстная операция в сессии, поэтому ее промежуточное состояние может быть сохранено и восстановлено функциями `C_GetOperationState`, `C_SetOperationState`. Данная возможность весьма полезна для поддержки протокола рукопожатия (handshake) в SSL/TLS.

Заметим, что ключ, используемый в контексте генерации имитовставки, рассматривается, как ключ аутентификации, а не как ключ шифрования, и поэтому задается в последнем параметре функции `C_SetOperationState`.

Размер состояния контекста генерации имитовставки зависит от реализации и в разных версиях библиотеки может оказаться различным, поэтому не следует надеяться на фиксированный размер буфера при сохранении этого состояния с помощью функции `C_GetOperationState`.

Листинг 3.21: `ckm_gost28147_mac.c`

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost28147_mac(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;
```

```
rc = funcs->C_OpenSession( SlotId ,
    CKF_RW_SESSION | CKF_SERIAL_SESSION,
    NULL_PTR, NULL_PTR, &hSession);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
    goto out;
}
fprintf(stderr , "C_OpenSession success\n");
/*
// log in as normal user
rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
    user_pin));
if (rc != CKR_OK) {
    fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
        rc);
    goto out_close;
}
fprintf(stderr, "C_Login success\n");
*/
rc = funcs->C_GetMechanismInfo( SlotId , CKM_GOST28147_MAC, &
    minfo);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "\n==== Mechanism CKM_GOST28147_MAC not supported
        =====\n");
} else {
    fprintf(stderr ,
        "\n==== CKM_GOST28147_MAC test
        =====\n");
    rc = test_gost28147_mac(hSession);
    if (rc != CKR_OK) {
        fprintf(stderr ,
            "ERROR CKM_GOST28147_MAC failed, rc = 0x%x\n", rc);
    } else {
        fprintf(stderr , "CKM_GOST28147_MAC test passed.\n");
    }
}

if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr ,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
```



```

else {
    fprintf(stderr, "C_CloseSession success\n");
}

out:
return rc;
}

CK_RV test_gost28147_mac(CK_SESSION_HANDLE sess)
{
    CK_RV rc = CKR_OK;
    CK_BYTE value[128];
    CK_ULONG len;
    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc = {CKM_GOST28147_MAC, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_KEY_TYPE key_type = CKK_GOST28147;
    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;

    static CK_BYTE key[] = {
        0xB2, 0x4C, 0x23, 0x32, 0xE5, 0x97, 0xAD, 0x26,
        0x39, 0xB9, 0x1A, 0xD4, 0xDF, 0x4E, 0x40, 0x61,
        0xAA, 0x38, 0xDD, 0xFE, 0x32, 0xE9, 0xD3, 0xED,
        0x4A, 0xC6, 0xEE, 0x08, 0x57, 0x5A, 0x6A, 0xAB
    };
    static CK_BYTE data[64];
    static CK_BYTE key2[] = {
        0xc3, 0x73, 0x0c, 0x5c, 0xbc, 0xca, 0xcf, 0x91,
        0x5a, 0xc2, 0x92, 0x67, 0x6f, 0x21, 0xe8, 0xbd,
        0x4e, 0xf7, 0x53, 0x31, 0xd9, 0x40, 0x5e, 0x5f,
        0x1a, 0x61, 0xdc, 0x31, 0x30, 0xa6, 0x50, 0x11
    };
    static CK_BYTE data2[2697];
    // CryptoPro gost28147 B Param Set
    static CK_BYTE oid1[] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02
    };
    static CK_BYTE et1[] = { 0xf6, 0x6a, 0x39, 0xde };
    static CK_ATTRIBUTE key_template[] = {
        { CKA_VALUE, key, sizeof(key) },
        { CKA_CLASS, &oclass, sizeof(oclass) },

```

```

    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_GOST28147PARAMS, oid1, sizeof(oid1) },
    { CKA_SIGN, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VERIFY, &ltrue, sizeof(CK_BBOOL) },
};
static CK_BYTE ivec[8] = {0};

// S-Terra CSP plug-in test data
static CK_BYTE data5[] = {
    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
};
static CK_BYTE key5[] = {
    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
};
// CryptoPro gost28147 A Param Set (default)
static CK_BYTE oid5[] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
};
static CK_BYTE ivec5[] = {
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37
};
static CK_ATTRIBUTE key_template5[] = {
    { CKA_VALUE, key5, sizeof(key5) },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_GOST28147PARAMS, oid5, sizeof(oid5) },
    { CKA_SIGN, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VERIFY, &ltrue, sizeof(CK_BBOOL) },
};
static CK_BYTE et5[] = {
//    0x77, 0xb9, 0x86, 0xba // with zero iv
    0xa3, 0xcd, 0x56, 0x64
};

static CK_BYTE data_short[] = {
    0x12, 0x00, 0xae, 0x6d, 0xc5, 0x87, 0xb0, 0x3f,
};
static CK_BYTE et_short[] = { 0xa5, 0x66, 0x40, 0x04 };

```

```
static CK_BYTE et_short_4 [] = { 0xd2, 0xbc, 0xfb, 0xe7 };
CK_BYTE *state = NULL;
CK_ULONG state_len = 0;

CK_BYTE key_to_wrap [] = {
0x58, 0x4c, 0x3f, 0xce, 0x56, 0xb3, 0x96, 0xd0,
0x8c, 0xf1, 0x5f, 0x60, 0xfc, 0x84, 0xaf, 0x6d,
0xbf, 0x1e, 0x6c, 0x6d, 0xa7, 0x9a, 0xd7, 0x4f,
0xca, 0x0e, 0x5c, 0x9d, 0x3d, 0xb2, 0xa2, 0xb9,
};

CK_BYTE wrapping_iv [] = {
0x56, 0xfa, 0x7d, 0x54, 0xb2, 0x90, 0x2c, 0x41,
};

CK_BYTE wrapping_key [] = {
0x3c, 0x34, 0x7c, 0xe7, 0xd5, 0x60, 0x83, 0x3a,
0xb4, 0x8f, 0xd5, 0xb0, 0xec, 0x6d, 0xd7, 0x29,
0x63, 0x9f, 0x64, 0x95, 0xb5, 0xdb, 0x3e, 0x12,
0x9e, 0x77, 0x0f, 0xe3, 0x40, 0xce, 0xcd, 0x3a,
};

CK_BYTE wrapped_key [] = {
0xdb, 0x46, 0x42, 0x7a, 0x12, 0x77, 0xd6, 0xdd,
0x8a, 0xd6, 0x35, 0x2f, 0x46, 0x98, 0x09, 0x9b,
0x60, 0xb3, 0x40, 0x58, 0x2b, 0xce, 0x1a, 0x99,
0x72, 0xda, 0x6c, 0xa2, 0x4c, 0xf0, 0xc1, 0xc7,
};

CK_BYTE wrapped_mac [] = {
0x36, 0xb3, 0x73, 0x20,
};

CK_ATTRIBUTE wrapping_key_template [] = {
{ CKA_VALUE, wrapping_key, sizeof(wrapping_key) },
{ CKA_CLASS, &oclass, sizeof(oclass) },
{ CKA_KEY_TYPE, &key_type, sizeof(key_type) },
{ CKA_GOST28147PARAMS, oid5, sizeof(oid5) },
{ CKA_SIGN, &ltrue, sizeof(CK_BBOOL) },
{ CKA_VERIFY, &ltrue, sizeof(CK_BBOOL) },
};
CK_CHAR str[4096];

keyh = CK_INVALID_HANDLE;
```

```
print_bytes(key, sizeof(key), str);
fprintf(stderr, "key:\n%s\n", str);

rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
memset(data, 0xd4, sizeof(data));
print_bytes(data, sizeof(data), str);
fprintf(stderr, "data:\n%s\n", str);
rc = funcs->C_Sign(sess, data, sizeof(data), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}
print_bytes(et1, sizeof(et1), str);
fprintf(stderr, "et1 MAC:\n%s\n", str);
print_bytes(value, len, str);
fprintf(stderr, "value MAC:\n%s\n", str);

CHECK(value, len, et1);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    goto end;
}

keyh = CK_INVALID_HANDLE;

rc = funcs->C_CreateObject(sess,
    key_template5, sizeof(key_template5)/sizeof(CK_ATTRIBUTE),
    &keyh);
```

```
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

mechanism->pParameter = ivec5;
mechanism->ulParameterLen = sizeof(ivec5);

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Sign(sess, data5, sizeof(data5), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et5);

len = sizeof(value);
rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}
rc = funcs->C_SignUpdate(sess, data5, 5);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignUpdate failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_GetOperationState(sess, NULL, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
    goto end;
}
state = (CK_BYTE *)malloc(state_len);
rc = funcs->C_GetOperationState(sess, state, &state_len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetOperationState failed: 0x%x\n", rc);
}
```

```
    goto end;
}
rc = funcs->C_SetOperationState(sess, state, state_len,
    CK_INVALID_HANDLE, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SetOperationState failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignUpdate(sess, data5+5, sizeof(data5)-5);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignUpdate failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignFinal(sess, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignFinal failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et5);

rc = funcs->C_SetOperationState(sess, state, state_len,
    CK_INVALID_HANDLE, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SetOperationState failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignUpdate(sess, data5+5, sizeof(data5)-5);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignUpdate failed: 0x%x\n", rc);
    goto end;
}

rc = funcs->C_SignFinal(sess, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignFinal failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et5);
```

```
free(state);
state = NULL;

mechanism->pParameter = NULL;
mechanism->ulParameterLen = 0;

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Sign(sess,
    data_short, sizeof(data_short), value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et_short);

rc = funcs->C_SignInit(sess, mechanism, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_SignInit failed: 0x%x\n", rc);
    goto end;
}

len = sizeof(value);
rc = funcs->C_Sign(sess,
    data_short, sizeof(data_short)-4, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_Sign failed: 0x%x\n", rc);
    goto end;
}

CHECK(value, len, et_short_4);

end:
if (keyh != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess, keyh);
}

return rc;
```

```
}
```

3.3.13 PKCS#8 и вывод открытого ключа по закрытому

Некоторые дополнительные механизмы LS11SW не определены в стандарте PKCS#11, а добавлены, исходя из потребностей решения некоторых практических задач. В данном примере закрытый ключ шифруется на пароле и упаковывается в структуру PKCS#8 с помощью дополнительного механизма CKM_GOST28147_PKCS8_KEY_WRAP. Заодно демонстрируется возможность вывода открытого ключа по закрытому другим дополнительным механизмом - CKM_GOSTR3410_PUBLIC_KEY_DERIVE. Заметим, что в стандарте PKCS#11 отсутствует возможность получить открытый ключ по закрытому, хотя в алгоритме ГОСТ Р34.10-2001 именно так и создается ключевая пара. В некоторых прикладных задачах вывод открытого ключа по закрытому все же приходится делать, потому что во входных данных присутствует только закрытый ключ.

Листинг 3.22: ckm_gost28147_pkcs8_key_wrap.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost28147_pkcs8_key_wrap(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;
```



```
printf(
"CKM_GOST28147_PKCS8_KEY_WRAP "
"and CKM_GOSTR3410_PUBLIC_KEY_DERIVE test\n");

rc = funcs->C_GetMechanismInfo(SlotId ,
    CKM_GOSTR3410_KEY_PAIR_GEN, &minfo);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "Mechanism CKM_GOSTR3410_KEY_PAIR_GEN not supported\n");
    return rc;
}
rc = funcs->C_GetMechanismInfo(SlotId ,
    CKM_GOST28147_PKCS8_KEY_WRAP, &minfo);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "Mechanism CKM_GOST28147_PKCS8_KEY_WRAP not supported\n");
    return rc;
}
rc = funcs->C_GetMechanismInfo(SlotId ,
    CKM_GOSTR3410_PUBLIC_KEY_DERIVE, &minfo);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "Mechanism CKM_GOSTR3410_PUBLIC_KEY_DERIVE not supported\n
");
    return rc;
}

rc = funcs->C_OpenSession(SlotId ,
    CKF_RW_SESSION | CKF_SERIAL_SESSION,
    NULL_PTR, NULL_PTR, &hSession);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
    goto out;
}
/*
rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
    user_pin));
if (rc != CKR_OK) {
    fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n
", rc);
    goto out_close;
}
*/
```

```

*/
rc = test_gost28147_pkcs8_key_wrap(hSession);

if (rc != CKR_OK) {
    fprintf(stderr,
"CKM_GOST28147_PKCS8_KEY_WRAP and
CKM_GOSTR3410_PUBLIC_KEY_DERIVE "
"test failed, rc = 0x%x\n",
rc);
} else {
    printf(
"CKM_GOST28147_PKCS8_KEY_WRAP and
CKM_GOSTR3410_PUBLIC_KEY_DERIVE "
"test SUCCESS\n");
}

funcs->C_CloseSession(hSession);
out:
return rc;
}

CK_RV test_gost28147_pkcs8_key_wrap(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    char str[4096];
    CK_BYTE value[1024];
    CK_ULONG len;
    CK_OBJECT_HANDLE pub_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE priv_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE unw_priv_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE der_pub_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE cipher_key = CK_INVALID_HANDLE;
    CK_MECHANISM mechanism_desc =
        {CKM_GOST28147_PKCS8_KEY_WRAP, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;
    CK_MECHANISM mechanism_der_desc =
        {CKM_GOSTR3410_PUBLIC_KEY_DERIVE, NULL, 0};
    CK_MECHANISM_PTR mechanism_der = &mechanism_der_desc;
    CK_MECHANISM mechanism_gen_256_desc =
        {CKM_GOSTR3410_KEY_PAIR_GEN, NULL, 0};
    CK_MECHANISM_PTR mechanism_gen_256 = &mechanism_gen_256_desc;
    ;
    CK_MECHANISM mechanism_gen_512_desc =
        {CKM_GOSTR3410_512_KEY_PAIR_GEN, NULL, 0};

```

```

    CK_MECHANISM_PTR mechanism_gen_512 = &mechanism_gen_512_desc
    ;

    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    static const CK_OBJECT_CLASS oclass_pub = CKO_PUBLIC_KEY;
    static const CK_OBJECT_CLASS oclass_priv = CKO_PRIVATE_KEY;
    static CK_KEY_TYPE key_type_256 = CKK_GOSTR3410;
    static CK_KEY_TYPE key_type_512 = CKK_GOSTR3410_512;
    // PAR 256 A OID
    static CK_BYTE gostR3410_256_params_A [] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01
    };
    // PAR 512 A OID
    CK_BYTE gostR3410_512_params_A [] = {
        0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x2, 0x01, 0x02, 0
        x01
    };
    // PAR HASH 1 OID
    static CK_BYTE gostR3411_94_params [] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
    };
    // PAR 3411-2012-256 OID
    CK_BYTE gostR3411_2012_256_params [] = {
        0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02
    };
    // PAR 3411-2012-512 OID
    CK_BYTE gostR3411_2012_512_params [] = {
        0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03
    };
    // PAR CIPHER A OID
    static CK_BYTE gost28147params_A [] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
    };
    /*
typedef struct CK_GOST28147_PKCS8_KEY_WRAP_PARAMS {
    CK_CHAR_PTR      pPassword;
    CK_ULONG         ulPasswordLen;
    CK_BYTE_PTR      pHashParOID;
    CK_ULONG         ulHashParOIDLen;
    CK_BYTE_PTR      pSalt;
    CK_ULONG         ulSaltLen;
    CK_ULONG         ulIterCount;
    CK_BYTE_PTR      pCipherParOID;

```

```

CK_ULONG          ulCipherParOIDLen;
CK_BYTE_PTR       pIV;
CK_ULONG          ulIVLen;
} CK_GOST28147_PKCS8_KEY_WRAP_PARAMS;
*/
CK_GOST28147_PKCS8_KEY_WRAP_PARAMS params;

static CK_BYTE salt [] = {
    0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0x26
};
static CK_ULONG iter = 2048;
static CK_BYTE iv [] = {
    0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0xab
};
// Use UTF-8 password
static CK_UTF8CHAR_PTR password = "4444";
static CK_ULONG password_len = 4;
CK_ATTRIBUTE pub_2001_256_template [] = {
    { CKA_GOSTR3410PARAMS,
      gostR3410_256_params_A, sizeof(gostR3410_256_params_A) },
    { CKA_GOSTR3411PARAMS,
      gostR3411_94_params, sizeof(gostR3411_94_params) },
};
CK_ATTRIBUTE pub_2012_256_template [] = {
    { CKA_GOSTR3410PARAMS,
      gostR3410_256_params_A, sizeof(gostR3410_256_params_A) },
    { CKA_GOSTR3411PARAMS,
      gostR3411_2012_256_params, sizeof(gostR3411_2012_256_params)
    },
};
CK_ATTRIBUTE pub_2012_512_template [] = {
    { CKA_GOSTR3410PARAMS,
      gostR3410_512_params_A, sizeof(gostR3410_512_params_A) },
    { CKA_GOSTR3411PARAMS,
      gostR3411_2012_512_params, sizeof(gostR3411_2012_512_params)
    },
};
// Template for token private key generation.
CK_ATTRIBUTE priv_template [] = {
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) },
    { CKA_SENSITIVE, &ltrue, sizeof(ltrue) }
};
// Additional attributes for token unwrapped private key
generation.

```

```

CK_ATTRIBUTE unw_256_template [] = {
    { CKA_CLASS, (CK_VOID_PTR)&oclass_priv, sizeof(oclass_priv)
    },
    { CKA_KEY_TYPE, &key_type_256, sizeof(key_type_256) }
};
CK_ATTRIBUTE unw_512_template [] = {
    { CKA_CLASS, (CK_VOID_PTR)&oclass_priv, sizeof(oclass_priv)
    },
    { CKA_KEY_TYPE, &key_type_512, sizeof(key_type_512) }
};
CK_ATTRIBUTE pub_2001_256_der_template [] = {
    { CKA_CLASS, (CK_VOID_PTR)&oclass_pub, sizeof(oclass_pub)
    },
    { CKA_KEY_TYPE, &key_type_256, sizeof(key_type_256) },
    { CKA_GOSTR3410PARAMS,
    gostR3410_256_params_A, sizeof(gostR3410_256_params_A) },
    { CKA_GOSTR3411PARAMS,
    gostR3411_94_params, sizeof(gostR3411_94_params) },
};
CK_ATTRIBUTE pub_2012_256_der_template [] = {
    { CKA_CLASS, (CK_VOID_PTR)&oclass_pub, sizeof(oclass_pub)
    },
    { CKA_KEY_TYPE, &key_type_256, sizeof(key_type_256) },
    { CKA_GOSTR3410PARAMS,
    gostR3410_256_params_A, sizeof(gostR3410_256_params_A) },
    { CKA_GOSTR3411PARAMS,
    gostR3411_2012_256_params, sizeof(gostR3411_2012_256_params)
    },
};
CK_ATTRIBUTE pub_2012_512_der_template [] = {
    { CKA_CLASS, (CK_VOID_PTR)&oclass_pub, sizeof(oclass_pub)
    },
    { CKA_KEY_TYPE, &key_type_512, sizeof(key_type_512) },
    { CKA_GOSTR3410PARAMS,
    gostR3410_512_params_A, sizeof(gostR3410_512_params_A) },
    { CKA_GOSTR3411PARAMS,
    gostR3411_2012_512_params, sizeof(gostR3411_2012_512_params)
    },
};
// При упаковке на выходе должна получиться структура такого в
// ида:
static CK_BYTE pr_key_bag [] = {
// 0
    0x30, 0x81, 0xa7,

```

```
// 3
    0x30, 0x5c,
// 5
    0x06, 0x09,
    0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x05, 0x0d,
// 16
    0x30, 0x4f,
// 18
    0x30, 0x2e,
// 20
    0x06, 0x09,
    0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x05, 0x0c,
// 31
    0x30, 0x21,
// 33
    0x04, 0x08,
// 35 - salt
    0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0
    x26,
// 43
    0x02, 0x02,
// 45 - iter (2048)
    0x08, 0x00,
// 47
    0x30, 0x11,
// 49
    0x06, 0x06,
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x0a,
// 57 - oid_3411_par
    0x06, 0x07,
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01,
// 66
    0x30, 0x1d,
// 68
    0x06, 0x06,
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x15,
// 76
    0x30, 0x13,
// 78
    0x04, 0x08,
// 80 - iv
    0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34,
    0xab,
// 88 - oid_28147_par
```

```

        0x06, 0x07,
        0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
// 97
        0x04, 0x47,
// 99 – encr_pr_key
        0x46, 0x5c, 0xc8, 0x21, 0xcd, 0xcc, 0xb8, 0x99,
        0x53, 0xa7, 0x72, 0xda, 0x20, 0x0f, 0x3d, 0xd5,
        0xab, 0x59, 0x11, 0x6f, 0x4f, 0x8a, 0x75, 0x9b,
        0xf4, 0xd1, 0x91, 0x7e, 0x9d, 0x2f, 0x79, 0xab,
        0x95, 0xb8, 0x54, 0xe2, 0x5b, 0x21, 0x61, 0xa8,
        0xe7, 0x2b, 0x58, 0x5b, 0xe2, 0x10, 0xd0, 0xd8,
        0xbb, 0xd4, 0x04, 0xdc, 0x06, 0x4c, 0x60, 0x14,
        0x60, 0x12, 0xd1, 0xf1, 0xb2, 0xbf, 0x39, 0xf7,
        0xc8, 0x35, 0x16, 0xc4, 0xea, 0xe9, 0xb6
// 170
    };
// Заметим, что при распаковке во входной структуре допускаетс
я
// использование 0x05, 0x00 вместо oid_3411_par. Соответственн
о,
// длина такой укороченной структуры будет меньше на 7 байтов.

// GOSTR3410–2001
rc = funcs->C_GenerateKeyPair(sess, mechanism_gen_256,
    pub_2001_256_template,
    sizeof(pub_2001_256_template)/sizeof(CK_ATTRIBUTE),
    priv_template, sizeof(priv_template)/sizeof(CK_ATTRIBUTE),
    &pub_key, &priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GenerateKeyPair failed, rc = 0x%x\n", rc)
    ;
    goto end;
}

params.pPassword = password;
params.ulPasswordLen = strlen(password);
params.pHashParOID = gostR3411_94_params;
params.ulHashParOIDLen = sizeof(gostR3411_94_params);
params.pSalt = salt;
    params.ulSaltLen = sizeof(salt);
    params.ulIterCount = iter;
params.pCipherParOID = gost28147params_A;
params.ulCipherParOIDLen = sizeof(gost28147params_A);
params.pIV = iv;

```

```

    params.ulIVLen = sizeof(iv);

    mechanism->pParameter = &params;
    mechanism->ulParameterLen = sizeof(params);
    rc = funcs->C_WrapKey(sess, mechanism,
        CK_INVALID_HANDLE, priv_key, value, &len);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_WrapKey failed, rc = 0x%x\n", rc);
        goto end;
    }

    print_bytes(value, len, str);
    printf("PKCS#8:\n%s\n", str);
/*
0x30, 0x81, 0xa7,
0x30, 0x5c,
0x06, 0x09,
0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x05, 0x0d,
0x30, 0x4f,
0x30, 0x2e,
0x06, 0x09,
0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x05, 0x0c
,
0x30, 0x21,
0x04, 0x08,
0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0x26,
0x02, 0x02,
0x08, 0x00,
0x30, 0x11,
0x06, 0x06,
0x2a, 0x85, 0x03, 0x02, 0x02, 0x0a,
0x06, 0x07,
0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01,
0x30, 0x1d,
0x06, 0x06,
0x2a, 0x85, 0x03, 0x02, 0x02, 0x15,
0x30, 0x13,
0x04, 0x08,
0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0xab,
0x06, 0x07,
0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
0x04, 0x47,
0x95, 0x44, 0x39, 0xe9, 0xce, 0xb5, 0x56, 0x29,
0xfa, 0x08, 0xf2, 0xab, 0x09, 0x81, 0x44, 0xca,

```



```

0xe0, 0x56, 0xc7, 0x60, 0xe2, 0x5c, 0xbd, 0x3c,
0xdd, 0x1c, 0x13, 0xf8, 0xf1, 0xe8, 0xd3, 0xf4,
0x98, 0xf0, 0x2e, 0x3c, 0xa0, 0x7d, 0xfe, 0x35,
0xee, 0xcf, 0xc4, 0xaf, 0xad, 0x93, 0x89, 0x48,
0xe5, 0xf2, 0x25, 0xbf, 0xe7, 0x61, 0x01, 0x8f,
0x50, 0x62, 0x08, 0x9b, 0xad, 0x9e, 0xcb, 0x42,
0xce, 0x17, 0x81, 0xc4, 0xea, 0xe9, 0xb6,
*/
mechanism->pParameter = password;
mechanism->ulParameterLen = strlen(password);
rc = funcs->C_UnwrapKey(sess, mechanism, CK_INVALID_HANDLE,
    value, len,
    unw_256_template, sizeof(unw_256_template)/sizeof(
    CK_ATTRIBUTE),
    &unw_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_UnwrapKey failed, rc = 0x%x\n", rc);
    goto end;
}
// Получаем открытый ключ по закрытому
rc = funcs->C_DeriveKey(sess, mechanism_der, unw_priv_key,
    pub_2001_256_der_template,
    sizeof(pub_2001_256_der_template)/sizeof(CK_ATTRIBUTE),
    &der_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed, rc = 0x%x\n", rc);
    goto end;
}
// Для проверки распакованного ключа нужно что-нибудь им под
// и проверить подпись открытым ключом.
{
    CK_MECHANISM mechanism_sign = {CKM_GOSTR3410, NULL, 0};
    // Подписываемый дайджест (32 байта)
    CK_BYTE pData[32] = {
        0x4D, 0x89, 0x9E, 0x48, 0xC5, 0x39, 0x64, 0xD1,
        0x78, 0xB4, 0x6D, 0x58, 0x40, 0x5F, 0x62, 0x8F,
        0xA4, 0x46, 0x4C, 0xDC, 0x73, 0x75, 0xB0, 0xE5,
        0x2E, 0x91, 0xFB, 0x64, 0x9C, 0x06, 0xAB, 0x75
    };
    CK_ULONG ulDataLen = 32;
    CK_BYTE pSignatureData[64];
    CK_ULONG signatureDataLen = 64;

```

```
        rc = funcs->C_SignInit(sess , &mechanism_sign ,
unw_priv_key);
    if (rc != CKR_OK) {
        fprintf(stderr , "C_SignInit failed, rc = 0x%x\n" , rc);
        goto end;
    }
        rc = funcs->C_Sign(sess ,
pData, ulDataLen, pSignatureData , &signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr , "C_Sign failed, rc = 0x%x\n" , rc);
        goto end;
    }

        rc = funcs->C_VerifyInit(sess , &mechanism_sign , pub_key)
;
    if (rc != CKR_OK) {
        fprintf(stderr , "C_VerifyInit failed, rc = 0x%x\n" , rc);
        goto end;
    }
        rc = funcs->C_Verify(sess ,
pData, ulDataLen, pSignatureData , signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr , "C_Verify failed, rc = 0x%x\n" , rc);
        goto end;
    }
}

        rc = funcs->C_VerifyInit(sess , &mechanism_sign ,
der_pub_key);
    if (rc != CKR_OK) {
        fprintf(stderr , "C_VerifyInit failed, rc = 0x%x\n" , rc);
        goto end;
    }
}
        rc = funcs->C_Verify(sess ,
pData, ulDataLen, pSignatureData , signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr , "C_Verify failed, rc = 0x%x\n" , rc);
        goto end;
    }
}
}
funcs->C_DestroyObject(sess , der_pub_key);
funcs->C_DestroyObject(sess , unw_priv_key);
funcs->C_DestroyObject(sess , pub_key);
funcs->C_DestroyObject(sess , priv_key);
```

```

// GOSTR3410-2012-256
rc = funcs->C_GenerateKeyPair(sess, mechanism_gen_256,
    pub_2012_256_template,
    sizeof(pub_2012_256_template)/sizeof(CK_ATTRIBUTE),
    priv_template, sizeof(priv_template)/sizeof(CK_ATTRIBUTE),
    &pub_key, &priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GenerateKeyPair failed, rc = 0x%x\n", rc)
    ;
    goto end;
}

params.pPassword = password;
params.ulPasswordLen = strlen(password);
params.pHashParOID = gostR3411_2012_256_params;
params.ulHashParOIDLen = sizeof(gostR3411_2012_256_params);
params.pSalt = salt;
    params.ulSaltLen = sizeof(salt);
    params.ulIterCount = iter;
params.pCipherParOID = gost28147params_A;
params.ulCipherParOIDLen = sizeof(gost28147params_A);
    params.pIV = iv;
    params.ulIVLen = sizeof(iv);

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);
rc = funcs->C_WrapKey(sess, mechanism,
    CK_INVALID_HANDLE, priv_key, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_WrapKey failed, rc = 0x%x\n", rc);
    goto end;
}

print_bytes(value, len, str);
printf("PKCS#8:\n%s\n", str);
/*
0x30, 0x81, 0xab,
    0x30, 0x5d,
        0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x05
    , 0x0d,
        0x30, 0x50,
            0x30, 0x2f,
                0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01,
0x05, 0x0c,

```

```

    0x30, 0x22,
    0x04, 0x08,
    0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0x26,
    0x02, 0x02,
    0x08, 0x00,
    0x30, 0x12,
    0x06, 0x06, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x0a,
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0
x02, 0x02,
    0x30, 0x1d,
    0x06, 0x06, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x15,
    0x30, 0x13,
    0x04, 0x08,
    0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0xab,
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
,
    0x04, 0x4a,
    0x68, 0x7f, 0x0d, 0x40, 0xbc, 0x97, 0xe6, 0x83,
    0x83, 0xdf, 0x4b, 0x9e, 0xff, 0x60, 0x64, 0xbd,
    0xb3, 0xea, 0xfe, 0x6b, 0xb2, 0x00, 0xc7, 0x78,
    0x80, 0x13, 0xc7, 0x61, 0x45, 0x76, 0x30, 0x53,
    0x5d, 0xf1, 0xcf, 0xdc, 0xf3, 0xd9, 0xdd, 0x50,
    0x7d, 0x7a, 0x2c, 0x87, 0x06, 0x2f, 0x28, 0xdb,
    0x1e, 0x94, 0xe3, 0xab, 0x9f, 0x0a, 0x75, 0x09,
    0xd3, 0xf9, 0x2e, 0xbe, 0xc6, 0xb1, 0xf4, 0x9c,
    0x92, 0xc9, 0xa4, 0x35, 0xd6, 0x6c, 0x25, 0xc7,
    0xbc, 0xcf,
*/
    mechanism->pParameter = password;
    mechanism->ulParameterLen = strlen(password);
    rc = funcs->C_UnwrapKey(sess, mechanism, CK_INVALID_HANDLE,
        value, len,
        unw_256_template,
        sizeof(unw_256_template)/sizeof(CK_ATTRIBUTE),
        &unw_priv_key);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_UnwrapKey failed, rc = 0x%x\n", rc);
        goto end;
    }
    // Получаем открытый ключ по закрытому
    rc = funcs->C_DeriveKey(sess, mechanism_der, unw_priv_key,
        pub_2012_256_der_template,
        sizeof(pub_2012_256_der_template)/sizeof(CK_ATTRIBUTE),
        &der_pub_key);

```

```

if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed, rc = 0x%x\n", rc);
    goto end;
}
// Для проверки распакованного ключа нужно что-нибудь им под
// писать
// и проверить подпись открытым ключом.
{
    CK_MECHANISM mechanism_sign = {CKM_GOSTR3410, NULL, 0};
// Подписываемый дайджест (32 байта)
    CK_BYTE pData[32] = {
        0x4D, 0x89, 0x9E, 0x48, 0xC5, 0x39, 0x64, 0xD1,
        0x78, 0xB4, 0x6D, 0x58, 0x40, 0x5F, 0x62, 0x8F,
        0xA4, 0x46, 0x4C, 0xDC, 0x73, 0x75, 0xB0, 0xE5,
        0x2E, 0x91, 0xFB, 0x64, 0x9C, 0x06, 0xAB, 0x75
    };
    CK_ULONG ulDataLen = 32;
    CK_BYTE pSignatureData[64];
    CK_ULONG signatureDataLen = 64;

    rc = funcs->C_SignInit(sess, &mechanism_sign,
unw_priv_key);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_SignInit failed, rc = 0x%x\n", rc);
        goto end;
    }
    rc = funcs->C_Sign(sess,
        pData, ulDataLen, pSignatureData, &signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Sign failed, rc = 0x%x\n", rc);
        goto end;
    }
    rc = funcs->C_VerifyInit(sess, &mechanism_sign, pub_key)
;
    if (rc != CKR_OK) {
        fprintf(stderr, "C_VerifyInit failed, rc = 0x%x\n", rc);
        goto end;
    }
    rc = funcs->C_Verify(sess,
        pData, ulDataLen, pSignatureData, signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Verify failed, rc = 0x%x\n", rc);
        goto end;
    }
}

```

```
}

    rc = funcs->C_VerifyInit(sess, &mechanism_sign,
der_pub_key);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_VerifyInit failed, rc = 0x%x\n", rc);
        goto end;
    }
    rc = funcs->C_Verify(sess,
pData, ulDataLen, pSignatureData, signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Verify failed, rc = 0x%x\n", rc);
        goto end;
    }
}
funcs->C_DestroyObject(sess, der_pub_key);
funcs->C_DestroyObject(sess, unw_priv_key);
funcs->C_DestroyObject(sess, pub_key);
funcs->C_DestroyObject(sess, priv_key);

// GOSTR3410-2012-512
rc = funcs->C_GenerateKeyPair(sess, mechanism_gen_512,
pub_2012_512_template,
sizeof(pub_2012_512_template)/sizeof(CK_ATTRIBUTE),
priv_template, sizeof(priv_template)/sizeof(CK_ATTRIBUTE),
&pub_key, &priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GenerateKeyPair failed, rc = 0x%x\n", rc)
;
    goto end;
}

params.pPassword = password;
params.ulPasswordLen = strlen(password);
params.pHashParOID = gostR3411_2012_512_params;
params.ulHashParOIDLen = sizeof(gostR3411_2012_512_params);
params.pSalt = salt;
    params.ulSaltLen = sizeof(salt);
    params.ulIterCount = iter;
params.pCipherParOID = gost28147params_A;
params.ulCipherParOIDLen = sizeof(gost28147params_A);
params.pIV = iv;
params.ulIVLen = sizeof(iv);
```

```

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);
rc = funcs->C_WrapKey(sess, mechanism,
    CK_INVALID_HANDLE, priv_key, value, &len);
if (rc != CKR_OK) {
    fprintf(stderr, "C_WrapKey failed, rc = 0x%x\n", rc);
    goto end;
}

print_bytes(value, len, str);
printf("PKCS#8:\n%s\n", str);
/*
0x30, 0x81, 0xcd,
    0x30, 0x5d,
        0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x05
    , 0x0d,
        0x30, 0x50,
            0x30, 0x2f,
                0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01,
0x05, 0x0c,
            0x30, 0x22,
                0x04, 0x08,
                    0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0x26,
                    0x02, 0x02,
                    0x08, 0x00,
                0x30, 0x12,
                    0x06, 0x06, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x0a,
                    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02,
0x03,
                0x30, 0x1d,
                    0x06, 0x06, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x15,
                0x30, 0x13,
                    0x04, 0x08,
                        0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34, 0xab,
                        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
0x04, 0x6c,
                    0x68, 0x5d, 0x0d, 0x40, 0xbc, 0x97, 0xd8, 0x83,
                    0x73, 0x78, 0x9a, 0x75, 0x9a, 0xa1, 0x44, 0xf1,
                    0x80, 0x83, 0x05, 0x32, 0x1a, 0x83, 0x36, 0x03,
                    0x9e, 0x76, 0x41, 0xb8, 0x56, 0x10, 0x25, 0x6c,
                    0x65, 0x83, 0xa8, 0x46, 0xd5, 0x1b, 0xb0, 0x04,
                    0x41, 0x5c, 0x13, 0x91, 0x76, 0x1c, 0x85, 0x59,
                    0x6b, 0x1d, 0x8d, 0xc8, 0xde, 0x93, 0x2e, 0x22,
                    0x1c, 0x5c, 0xec, 0xaf, 0x1f, 0xb7, 0x7b, 0x63,

```

```

    0xa5, 0x72, 0x1f, 0x29, 0xd6, 0x07, 0xf6, 0xdf,
    0x8d, 0x69, 0xe4, 0x49, 0x35, 0xfb, 0x71, 0xda,
    0xb7, 0x4b, 0x17, 0x8c, 0xae, 0xbd, 0xcf, 0x62,
    0xd2, 0xe6, 0xdb, 0x4d, 0x7c, 0x47, 0x46, 0x82,
    0xdb, 0x0c, 0xcd, 0x5b, 0x0a, 0xc0, 0xd8, 0x42,
    0x58, 0x77, 0xe7, 0x7e,
*/
mechanism->pParameter = password;
mechanism->ulParameterLen = strlen(password);
rc = funcs->C_UnwrapKey(sess, mechanism, CK_INVALID_HANDLE,
    value, len,
    unw_512_template, sizeof(unw_512_template)/sizeof(
CK_ATTRIBUTE),
    &unw_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_UnwrapKey failed, rc = 0x%x\n", rc);
    goto end;
}
// Получаем открытый ключ по закрытому
rc = funcs->C_DeriveKey(sess, mechanism_der, unw_priv_key,
    pub_2012_512_der_template,
    sizeof(pub_2012_512_der_template)/sizeof(CK_ATTRIBUTE),
    &der_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed, rc = 0x%x\n", rc);
    goto end;
}
// Для проверки распакованного ключа нужно что-нибудь им под
писать
// и проверить подпись открытым ключом.
{
    CK_MECHANISM mechanism_sign = {CKM_GOSTR3410_512, NULL,
0};
// Подписываемый дайджест (64 байта)
    CK_BYTE pData[] = {
        0x4D, 0x89, 0x9E, 0x48, 0xC5, 0x39, 0x64, 0xD1,
        0x78, 0xB4, 0x6D, 0x58, 0x40, 0x5F, 0x62, 0x8F,
        0xA4, 0x46, 0x4C, 0xDC, 0x73, 0x75, 0xB0, 0xE5,
        0x2E, 0x91, 0xFB, 0x64, 0x9C, 0x06, 0xAB, 0x75,
        0x4D, 0x89, 0x9E, 0x48, 0xC5, 0x39, 0x64, 0xD1,
        0x78, 0xB4, 0x6D, 0x58, 0x40, 0x5F, 0x62, 0x8F,
        0xA4, 0x46, 0x4C, 0xDC, 0x73, 0x75, 0xB0, 0xE5,
        0x2E, 0x91, 0xFB, 0x64, 0x9C, 0x06, 0xAB, 0x75,
    };
};

```



```
    CK_ULONG ulDataLen = 64;
    CK_BYTE pSignatureData[128];
    CK_ULONG signatureDataLen = 128;

    rc = funcs->C_SignInit(sess, &mechanism_sign,
unw_priv_key);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_SignInit failed, rc = 0x%x\n", rc);
        goto end;
    }
    rc = funcs->C_Sign(sess,
pData, ulDataLen, pSignatureData, &signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Sign failed, rc = 0x%x\n", rc);
        goto end;
    }

    rc = funcs->C_VerifyInit(sess, &mechanism_sign, pub_key)
;
    if (rc != CKR_OK) {
        fprintf(stderr, "C_VerifyInit failed, rc = 0x%x\n", rc);
        goto end;
    }
    rc = funcs->C_Verify(sess,
pData, ulDataLen, pSignatureData, signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Verify failed, rc = 0x%x\n", rc);
        goto end;
    }

    rc = funcs->C_VerifyInit(sess, &mechanism_sign,
der_pub_key);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_VerifyInit failed, rc = 0x%x\n", rc);
        goto end;
    }
    rc = funcs->C_Verify(sess,
pData, ulDataLen, pSignatureData, signatureDataLen);
    if (rc != CKR_OK) {
        fprintf(stderr, "C_Verify failed, rc = 0x%x\n", rc);
        goto end;
    }
}
}
funcs->C_DestroyObject(sess, der_pub_key);
```

```
    funcs->C_DestroyObject(sess , unw_priv_key);
    funcs->C_DestroyObject(sess , pub_key);
    funcs->C_DestroyObject(sess , priv_key);

    printf("SUCCESS\n");
end:

    return rc;
}
```

3.3.14 Шифрование ключей по ГОСТ 28147-89

В данном примере секретный ключ шифруется на ключе согласования механизмом CKM_GOST28147_KEY_WRAP. Результатом шифрования является зашифрованный в режиме простой замены ключ, сопровождаемый имитовставкой исходного значения ключа (всего 36 байтов), как определено в [8] п.6. При этом, значение UKM передается механизму в качестве параметра, а в результирующую структуру не включается.

Листинг 3.23: ckm_gost28147_key_wrap.c

```
#include "test_common.h"

int main(int argc, char* argv[])
{
    CK_RV rvResult;
#ifdef WIN32
    HMODULE hPkcsLib = NULL;
    HMODULE hPkcsLib2 = NULL;
#else
    void *hPkcsLib = NULL;
    void *hPkcsLib2 = NULL;
#endif
    CK_C_GetFunctionList pcGetFunctionList = 0;
    CK_C_GetFunctionList pcGetFunctionList2 = 0;
    CK_FUNCTION_LIST_PTR Pkcs11FuncList = NULL;
    CK_FUNCTION_LIST_PTR Pkcs11FuncList2 = NULL;
    CK_SLOT_ID_PTR pSlotList = NULL;
    CK_SLOT_ID_PTR pSlotList2 = NULL;
    CK_SLOT_ID SlotId;
    CK_SLOT_ID SlotId2;
    CK_ULONG ulSlotCount;
    CK_ULONG ulSlotCount2;
    CK_ULONG j;
```

```

CK_SLOT_INFO      SlotInfo;
CK_SLOT_INFO      SlotInfo2;
CK_SESSION_HANDLE  hSessionSend, hSessionRecv;

CK_UTF8CHAR_PTR   pcUserPIN = (CK_UTF8CHAR_PTR)"01234567";
CK_ULONG          ulPinLength = 8;           // PIN length

CK_BYTE           pbPlainText [] =
    "This is plaintext for encryption and decryption";
CK_ULONG          ulPlainTextSize = sizeof(pbPlainText);

CK_BYTE_PTR       pbCipherText = NULL;
CK_ULONG          ulCipherSize = 0;

CK_BYTE_PTR       pbDecryptedText = NULL;
CK_ULONG          ulDecryptedSize = 0;

/*****/
CK_BBOOL          blTrue = CK_TRUE,
                 blFalse = CK_FALSE;
CK_ULONG          ulKeyType_Gost2001 = CKK_GOSTR3410,
                 ulKeyType_Gost28147 = CKK_GOST28147,
                 ulKeyType_Gost3411 = CKK_GOSTR3411,
                 ulClass_PubKey = CKO_PUBLIC_KEY,
                 ulClass_PriKey = CKO_PRIVATE_KEY,
                 ulClass_SecKey = CKO_SECRET_KEY,
                 ulClass_Domain = CKO_DOMAIN_PARAMETERS;
// PAR ECC A OID
CK_BYTE          gostR3410params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01
};
// PAR HASH 1 OID
CK_BYTE          gostR3411params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
};
// PAR CIPHER A OID
CK_BYTE          gost28147params_A [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
};
// PAR CIPHER B OID
CK_BYTE          gost28147params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02
};
// template for GOST R 34.10-2001 public key

```

```

CK_ATTRIBUTE caGOST_PublicKeyTemplate [] =
{
    {CKA_TOKEN,          &blFalse ,          sizeof(CK_BBOOL)
    },
    {CKA_PRIVATE,       &blFalse ,          sizeof(CK_BBOOL)
    },
    {CKA_GOSTR3410_PARAMS, gostR3410params , sizeof(
gostR3410params )},
    {CKA_GOSTR3411_PARAMS, gostR3411params , sizeof(
gostR3411params )},
    {CKA_GOST28147_PARAMS, gost28147params_A , sizeof(
gost28147params_A )},
};
// template for GOST R 34.10–2001 private key
CK_ATTRIBUTE caGOST_PrivateKeyTemplate [] =
{
    {CKA_TOKEN,          &blFalse ,          sizeof(
CK_BBOOL) },
    {CKA_PRIVATE,       &blFalse ,          sizeof(
CK_BBOOL) },
    {CKA_DERIVE,        &blTrue ,           sizeof(
CK_BBOOL) },
//    {CKA_EXTRACTABLE,  &blTrue ,           sizeof(CK_BBOOL) },
};
// template for secret GOST key
CK_ATTRIBUTE caGOST_SecretKeyTemplate [] =
{
    {CKA_CLASS,         &ulClass_SecKey ,    sizeof(CK_ULONG) },
    {CKA_KEY_TYPE,     &ulKeyType_Gost28147 , sizeof(CK_ULONG) },
    {CKA_PRIVATE,      &blFalse ,           sizeof(CK_BBOOL) },
    {CKA_ENCRYPT,       &blTrue ,            sizeof(CK_BBOOL) },
    {CKA_DECRYPT,       &blTrue ,            sizeof(CK_BBOOL) },
    {CKA_WRAP,         &blTrue ,            sizeof(CK_BBOOL) },
    {CKA_EXTRACTABLE, &blTrue ,            sizeof(CK_BBOOL) },
    {CKA_GOST28147_PARAMS, gost28147params_A , sizeof(
gost28147params_A )},
};
// template for derive key
CK_ATTRIBUTE caDeriveKey [] =
{
    {CKA_CLASS,         &ulClass_SecKey ,    sizeof(CK_ULONG) },
    {CKA_KEY_TYPE,     &ulKeyType_Gost28147 , sizeof(CK_ULONG) },
    {CKA_ENCRYPT,       &blTrue ,            sizeof(CK_BBOOL) },
    {CKA_DECRYPT,       &blTrue ,            sizeof(CK_BBOOL) },
};

```

```

    {CKA_WRAP, &blTrue, sizeof(CK_BBOOL)},
    {CKA_UNWRAP, &blTrue, sizeof(CK_BBOOL)},
    {CKA_GOST28147_PARAMS, gost28147params_A, sizeof(
gost28147params_A) },
    };
CK_ULONG    ulPubKeyCount =
    sizeof(caGOST_PublicKeyTemplate)/sizeof(CK_ATTRIBUTE),
ulPriKeyCount =
    sizeof(caGOST_PrivateKeyTemplate)/sizeof(CK_ATTRIBUTE),
ulSecKeyCount =
    sizeof(caGOST_SecretKeyTemplate)/sizeof(CK_ATTRIBUTE),
ulDeriveKeyCount =
    sizeof(caDeriveKey)/sizeof(CK_ATTRIBUTE);
CK_OBJECT_HANDLE hKeyForWU = 0, // key which will be wrapped
and unwrapped

sender        hSendPubKey = 0, // handle to public key of
sender        hSendPriKey = 0, // handle to private key of
recipient    hRecpPubKey = 0, // handle to public key of
recipient    hRecpPriKey = 0, // handle to private key of
sender        hSendDH_Key = 0, // Diffy–Hellman key of the
recipient    hRecpDH_Key = 0, // Diffy–Hellman key of the
the sender   hUnwrappedSendKey = 0, // unwrapped key of
the recipient hUnwrappedRecpKey = 0; // unwrapped key of

CK_MECHANISM cmKeyGenMechanism, // mechanism for key pair
generation
              cmWrapMechanism, // mechanism for key wrap
/unwrap
              cmDeriveMechanism, // mechanism for key
derivation
              cmCryptMechanism; // mechanism for encrypt/
decrypt

CK_BYTE_PTR  pbWrappedKeySend; // wrapped key

CK_ULONG     ulWrappedKeyLen, // length of wrapped key

```

```

        i ;

CK_BYTE_PTR    pbSecretKeyParam = NULL,
                pbSend_PubKeyValue = NULL,
                pbRecp_PubKeyValue = NULL,
                pbSend_PriKeyValue = NULL,
                pbRecp_PriKeyValue = NULL,
                pbSecret_Key_Value = NULL,
                pbSendDH_Key_Value = NULL,
                pbRecpDH_Key_Value = NULL;

CK_BYTE_PTR    pbImitValue = NULL;
CK_ULONG       ulImitSize = 0;
CK_KEY_TYPE    keyType = 0;
CK_ATTRIBUTE   caSecretKeyParam = {CKA_GOST28147_PARAMS, pbSecretKeyParam,
0},
// template for value of sender public key
caSend_PubKeyValue = {CKA_VALUE, pbSend_PubKeyValue, 0},
caSend_PriKeyValue = {CKA_VALUE, pbSend_PriKeyValue, 0},
// template for value of recipient public key
caRecp_PubKeyValue = {CKA_VALUE, pbRecp_PubKeyValue, 0},
caRecp_PriKeyValue = {CKA_VALUE, pbRecp_PriKeyValue, 0},
caSecret_Key_Value = {CKA_VALUE, pbSecret_Key_Value, 0},
caKeyType = {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
SendDH_Key_Value = {CKA_VALUE, pbSendDH_Key_Value, 0},
RecpDH_Key_Value = {CKA_VALUE, pbRecpDH_Key_Value, 0};
// parameters for derivation mechanism
CK_GOSTR3410_DERIVE_PARAMS_PTR DeriveParams;
// UKM must be non-zero by RFC4357
CK_BYTE UKM[8] = {0x28, 0xaf, 0xc5, 0x50, 0x9d, 0x0c, 0x74, 0xb3};
CK_ULONG ulUKMLen = 8;
// parameters for wrapping mechanism
CK_BYTE pWrapIV[] = {
    0x56, 0xfa, 0x7d, 0x54, 0xb2, 0x90, 0x2c, 0x41
};
CK_BYTE iv[] = {
    0x37, 0x2a, 0x7f, 0x00, 0x2c, 0xea, 0x7d, 0x39
};

CK_CHAR *api_path = PKCS11_API_PATH;
CK_UTF8CHAR *user_pin = "01234567";
CK_ULONG slot_num = 0;
CK_CHAR *api_path2 = PKCS11_API_PATH;

```

```
CK_UTF8CHAR *user_pin2 = "01234567";
CK_ULONG slot_num2 = 0;

printf("Starting CKM_GOST28147_KEY_WRAP test\n");

for (i=1; i<(CK_ULONG) argc; i++) {
    if (strcmp("-api", argv[i]) == 0) {
        ++i;
        api_path = argv[i];
    } else if (strcmp("-slot", argv[i]) == 0) {
        ++i;
        slot_num = atoi(argv[i]);
    } else if (strcmp("-user_pin", argv[i]) == 0) {
        ++i;
        user_pin = argv[i];
    }
}

api_path2 = api_path;
user_pin2 = user_pin;
slot_num2 = slot_num;

#ifdef WIN32
    hPkcsLib = LoadLibrary(api_path);
#else
    hPkcsLib = dlopen(api_path, RTLD_NOW);
#endif
    if ( hPkcsLib == NULL ) {
        printf(
            "Can't load PKCS#11 API library. "
            "Check API library path.\n");
#ifdef WIN32
        printf("dlerror: %s\n", dlerror());
#endif
    }
    return -1;
}

#ifdef WIN32
    pcGetFunctionList =
        (CK_C_GetFunctionList)GetProcAddress(
            hPkcsLib, "C_GetFunctionList");
#else
    pcGetFunctionList =
        (CK_C_GetFunctionList)dlsym(
            hPkcsLib, "C_GetFunctionList");

```

```
#endif

// get PKCS #11 function list
rvResult = pcGetFunctionList(&Pkcs11FuncList);
printf("Load PKCS #11 function list result: 0x%x\n", rvResult)
;
if (rvResult != CKR_OK) return rvResult;

// initialize Cryptoki
rvResult = Pkcs11FuncList->C_Initialize(NULL);
printf("Initialize Cryptoki result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

// get slot list
rvResult = Pkcs11FuncList->C_GetSlotList(
    CK_FALSE, NULL, &ulSlotCount);
printf("Get slot list result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

if (ulSlotCount > 0)
{
    // allocate memory for slot list
    pSlotList = (CK_SLOT_ID_PTR) malloc(
        ulSlotCount * sizeof(CK_SLOT_ID));

    rvResult =
        Pkcs11FuncList->C_GetSlotList(CK_FALSE,
            pSlotList, &ulSlotCount);

    if (rvResult != CKR_OK) return rvResult;
    printf("Slot count: %d\n", ulSlotCount);
}
else return -3;

// get information about sender and recipient slots.
for(i=0; i<ulSlotCount; ++i)
{
    rvResult = Pkcs11FuncList->C_GetSlotInfo(pSlotList[i], &
        SlotInfo);
    if (rvResult == CKR_OK)
    {
        // if a token is present in this slot
        if ((SlotInfo.flags & CKF_TOKEN_PRESENT) ==
            CKF_TOKEN_PRESENT)
        {
            SlotId = pSlotList[i];
        }
    }
}
```



```

        break;
    }
}
}
if (i >= ulSlotCount) {
    printf("No slots with token present\n");
    return -3;
}
printf("Slot ID: sender %d\n", SlotId);

// load library PKCS #11
#ifdef WIN32
    hPkcsLib2 = LoadLibrary(api_path2);
#else
    hPkcsLib2 = dlopen(api_path2, RTLD_NOW);
#endif
if ( hPkcsLib2 == NULL ) {
    printf("Can't load PKCS#11 API library. "
        "Check API library path.\n");
#ifdef WIN32
    printf("dlerror: %s\n", dlerror());
#endif
    return FALSE;
}
#ifdef WIN32
    pcGetFunctionList2 =
        (CK_C_GetFunctionList)GetProcAddress(
            hPkcsLib2, "C_GetFunctionList");
#else
    pcGetFunctionList2 =
        (CK_C_GetFunctionList)dlsym(
            hPkcsLib2, "C_GetFunctionList");
#endif

// get PKCS #11 function list
rvResult = pcGetFunctionList2(&Pkcs11FuncList2);
printf("Load PKCS #11 function list result: 0x%x\n", rvResult)
;
if (rvResult != CKR_OK) return rvResult;

// initialize Cryptoki
rvResult = Pkcs11FuncList2->C_Initialize(NULL);
printf("Initialize Cryptoki result: 0x%x\n", rvResult);
if (rvResult != CKR_OK && rvResult

```

```
!= CKR_CRYPTOKI_ALREADY_INITIALIZED) return rvResult;

// get slot list
rvResult =
    Pkcs11FuncList2->C_GetSlotList(CK_TRUE, NULL, &ulSlotCount2)
;
printf("Get slot list result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

if (ulSlotCount2 > 0)
{
    // allocate memory for slot list
    pSlotList2 = (CK_SLOT_ID_PTR) malloc(
        ulSlotCount2 * sizeof(CK_SLOT_ID));

    rvResult =
        Pkcs11FuncList2->C_GetSlotList(
            CK_TRUE, pSlotList2, &ulSlotCount2);

    if (rvResult != CKR_OK) return rvResult;
    printf("Slot count: %d\n", ulSlotCount2);
}
else return -3;

// get information about sender and recipient slots.
j = 0;
for(i=0; i<ulSlotCount2; ++i)
{
    rvResult = Pkcs11FuncList2->C_GetSlotInfo(
        pSlotList2[i], &SlotInfo2);
    if (rvResult == CKR_OK)
    {
        // if a token is present in this slot
        if ((SlotInfo2.flags & CKF_TOKEN_PRESENT)
            == CKF_TOKEN_PRESENT)
        {
            SlotId2 = pSlotList2[i];
            break;
        }
    }
}
if (i >= ulSlotCount2) {
    printf("No slots with token present\n");
    return -3;
}
printf("Slot ID: recipient %d\n", SlotId2);
```

```

// open session for slot with ID = SlotId[0]
rvResult = Pkcs11FuncList->C_OpenSession( SlotId ,
    (CKF_SERIAL_SESSION | CKF_RW_SESSION) ,
    NULL,
    0,
    &hSessionSend );
printf("Sender open session result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;
// open session for slot with ID = SlotId[1]
rvResult = Pkcs11FuncList2->C_OpenSession( SlotId2 ,
    (CKF_SERIAL_SESSION | CKF_RW_SESSION) ,
    NULL,
    0,
    &hSessionRecp );
printf("Recipient open session result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

cmKeyGenMechanism.mechanism = CKM_GOST28147_KEY_GEN;
cmKeyGenMechanism.pParameter = NULL; //gost28147params_A ;
cmKeyGenMechanism.ulParameterLen = 0; //sizeof(
    gost28147params_A );

// generate secret key which will be wrapped and unwrapped
printf(
    "Sender generate key for wrapping and unwrapping (hKeyForWU)
    \n");
printf("    mechanism type: CKM_GOST28147_KEY_GEN\n");

rvResult = Pkcs11FuncList->C_GenerateKey( hSessionSend ,
    &cmKeyGenMechanism ,
    caGOST_SecretKeyTemplate ,
    ulSecKeyCount ,
    &hKeyForWU );
printf("    Sender generate key hKeyForWU "
    "for wrapping and unwrapping: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    key handle: %d\n", (unsigned long)hKeyForWU);
{
    CK_BYTE value[32];
    CK_ATTRIBUTE attr_get[] = {
        {CKA_VALUE, NULL, 0},

```

```

};
attr_get[0].pValue = value;
attr_get[0].ulValueLen = sizeof(value);
rvResult = Pkcs11FuncList->C_GetAttributeValue(
    hSessionSend, hKeyForWU,
    attr_get, sizeof(attr_get)/sizeof(CK_ATTRIBUTE));
printf("    key to wrap:\n");
print_hex(value, attr_get[0].ulValueLen);
}
printf("    wrapping IV:\n");
print_hex(pWrapIV, sizeof(pWrapIV));

// generate sender key pair with GOST R 34.10-2001
printf("Generate key pair of sender\n");
printf("    mechanism type: CKM_GOSTR3410_KEY_PAIR_GEN\n");

cmKeyGenMechanism.mechanism = CKM_GOSTR3410_KEY_PAIR_GEN;
cmKeyGenMechanism.pParameter = NULL;
cmKeyGenMechanism.ulParameterLen = 0;
rvResult = Pkcs11FuncList->C_GenerateKeyPair(hSessionSend,
    &cmKeyGenMechanism,
    caGOST_PublicKeyTemplate,
    ulPubKeyCount,
    caGOST_PrivateKeyTemplate,
    ulPriKeyCount,
    &hSendPubKey,
    &hSendPriKey);
printf("    generate sender key pair: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    sender public key handle: %d\n",
    (unsigned long)hSendPubKey);
printf("    sender private key handle: %d\n",
    (unsigned long)hSendPriKey);

// generate recipient key pair with GOST R 34.10-2001
printf("Generate key pair of recipient\n");
printf("    mechanism type: CKM_GR3410_KEY_PAIR_GEN\n");

rvResult = Pkcs11FuncList2->C_GenerateKeyPair(hSessionRecp,
    &cmKeyGenMechanism,
    caGOST_PublicKeyTemplate,
    ulPubKeyCount,

```

```
    caGOST_PrivateKeyTemplate ,
    ulPriKeyCount ,
    &hRecpPubKey ,
    &hRecpPriKey);
printf("    generate recipient key pair: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    recipient public key handle: %d\n",
    (unsigned long)hRecpPubKey);
printf("    recipient private key handle: %d\n",
    (unsigned long)hRecpPriKey);

// get value of sender public key
printf("Get value of sender public key\n");

rvResult = Pkcs11FuncList->C_GetAttributeValue( hSessionSend ,
    hSendPubKey ,
    &caSend_PubKeyValue ,
    1);
if (rvResult == CKR_OK)
{
    caSend_PubKeyValue.pValue =
        (CK_BYTE_PTR) malloc( caSend_PubKeyValue.ulValueLen );
    rvResult = Pkcs11FuncList->C_GetAttributeValue( hSessionSend ,
        hSendPubKey ,
        &caSend_PubKeyValue ,
        1);
}
printf("    Get sender public key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

pbSend_PubKeyValue = (CK_BYTE_PTR) caSend_PubKeyValue.pValue;
printf("    Sender public key value: \n");
print_hex(pbSend_PubKeyValue, caSend_PubKeyValue.ulValueLen);

// get value of recipient public key
printf("Get value of recipient public key\n");
rvResult = Pkcs11FuncList2->C_GetAttributeValue( hSessionRecp ,
    hRecpPubKey ,
    &caRecp_PubKeyValue ,
    1);
```

```
if (rvResult == CKR_OK)
{
    caRecp_PubKeyValue.pValue =
        (CK_BYTE_PTR) malloc(caRecp_PubKeyValue.ulValueLen);
    rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSessionRecp
,
    hRecpPubKey,
        &caRecp_PubKeyValue,
        1);
}

printf("    Get recipient public key value: result: 0x%x\n",
    rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    Recipient public key value: \n");
pbRecp_PubKeyValue = (CK_BYTE_PTR) caRecp_PubKeyValue.pValue;
print_hex(pbRecp_PubKeyValue, caRecp_PubKeyValue.ulValueLen);

// fill parameters for derivation mechanism
DeriveParams = (CK_GOSTR3410_DERIVE_PARAMS_PTR)
    malloc(sizeof(CK_GOSTR3410_DERIVE_PARAMS));
DeriveParams->kdf = CKD_CPDIVERSIFY_KDF;
DeriveParams->pPublicData = (CK_BYTE_PTR) caRecp_PubKeyValue.
    pValue;
DeriveParams->ulPublicDataLen = caRecp_PubKeyValue.ulValueLen;
DeriveParams->pUKM = UKM;
DeriveParams->ulUKMLen = ulUKMLen;

cmDeriveMechanism.mechanism = CKM_GOSTR3410_DERIVE;
cmDeriveMechanism.pParameter = DeriveParams;
cmDeriveMechanism.ulParameterLen = sizeof(
    CK_GOSTR3410_DERIVE_PARAMS);

printf("Derive Diffie-Hellman key\n");
printf("    derivation mechanism: CKM_GOSTR3410_DERIVE\n");

// derive Diffie-Hellman key of sender
printf("    derive Diffie-Hellman key for sender\n");
rvResult = Pkcs11FuncList->C_DeriveKey(hSessionSend,
    &cmDeriveMechanism,
    hSendPriKey,
    caDeriveKey,
    ulDeriveKeyCount,
```

```
    &hSendDH_Key);
printf("    derive key result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    sender's Diffie-Hellman key handle: %d\n",
    (unsigned long)hSendDH_Key);
{
    CK_BYTE value[32];
    CK_ATTRIBUTE attr_get[] = {
        {CKA_VALUE, NULL, 0},
    };
    attr_get[0].pValue = value;
    attr_get[0].ulValueLen = sizeof(value);
    rvResult = Pkcs11FuncList->C_GetAttributeValue(hSessionSend,
        hSendDH_Key,
        attr_get, sizeof(attr_get)/sizeof(CK_ATTRIBUTE));
    printf("    sender KEK:\n");
    print_hex(value, attr_get[0].ulValueLen);
}

// derive Diffie-Hellman key of recipient
printf("    derive Diffie-Hellman key for recipient\n");
DeriveParams->pPublicData = (CK_BYTE_PTR) caSend_PubKeyValue.
    pValue;
DeriveParams->ulPublicDataLen = caSend_PubKeyValue.ulValueLen;

rvResult = Pkcs11FuncList2->C_DeriveKey(hSessionRecp,
    &cmDeriveMechanism,
    hRecpPriKey,
    caDeriveKey,
    ulDeriveKeyCount,
    &hRecpDH_Key);
printf("    derive key result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    recipient's Diffie-Hellman key handle: %d\n",
    (unsigned long)hRecpDH_Key);
{
    CK_BYTE value[32];
    CK_ATTRIBUTE attr_get[] = {
        {CKA_VALUE, NULL, 0},
    };
    attr_get[0].pValue = value;
    attr_get[0].ulValueLen = sizeof(value);
}
```

```

    rvResult = Pkcs11FuncList2->C_GetAttributeValue(hSessionRecv
, hRecvDH_Key,
    attr_get, sizeof(attr_get)/sizeof(CK_ATTRIBUTE));
    printf("    recipient KEK:\n");
    print_hex(value, attr_get[0].ulValueLen);
}

// wrap key hKeyForWU with sender Diffie-Hellman key
hSendDH_Key
// Здесь используется УКМ в качестве IV в соответствии с RFC
4357
cmWrapMechanism.mechanism = CKM_GOST28147_KEY_WRAP;
cmWrapMechanism.pParameter = UKM;
cmWrapMechanism.ulParameterLen = ulUKMLen;

printf("Wrap key hKeyForWU with sender Diffie-Hellman key\n");

rvResult = Pkcs11FuncList->C_WrapKey(hSessionSend,
    &cmWrapMechanism,
    hSendDH_Key,
    hKeyForWU,
    NULL,
    &ulWrappedKeyLen);
if (rvResult == CKR_OK)
{
    pbWrappedKeySend = (CK_BYTE_PTR) malloc(ulWrappedKeyLen);
    rvResult = Pkcs11FuncList->C_WrapKey(hSessionSend,
        &cmWrapMechanism,
        hSendDH_Key,
        hKeyForWU,
        pbWrappedKeySend,
        &ulWrappedKeyLen);
}

printf("    wrap result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    wrapped key:\n");
print_hex(pbWrappedKeySend, ulWrappedKeyLen);

// unwrap wrapped key with recipient Diffie-Hellman key
printf("Unwrap key hUnwrappedRecvKey "
    "from pbWrappedKeySend with recipient Diffie-Hellman key\n")
;

```



```
rvResult = Pkcs11FuncList2->C_UnwrapKey(hSessionRecv ,
    &cmWrapMechanism,
    hRecvDH_Key,
    pbWrappedKeySend,
    ulWrappedKeyLen,
    caGOST_SecretKeyTemplate,
    ulSecKeyCount,
    &hUnwrappedRecvKey);
printf("    unwrap key: result: 0x%x\n", rvResult);
if (rvResult != CKR_OK) return rvResult;

printf("    key handle (hUnwrappedRecvKey): %d\n",
    (unsigned long)hUnwrappedRecvKey);
{
    CK_BYTE value[32];
    CK_ATTRIBUTE attr_get[] = {
        {CKA_VALUE, NULL, 0},
    };
    attr_get[0].pValue = value;
    attr_get[0].ulValueLen = sizeof(value);
    rvResult = Pkcs11FuncList2->C_GetAttributeValue(
        hSessionRecv, hUnwrappedRecvKey,
        attr_get, sizeof(attr_get)/sizeof(CK_ATTRIBUTE));
    printf("    recipient unvrapped key:\n");
    print_hex(value, attr_get[0].ulValueLen);
}

printf("Sender encrypt with hKeyForWU:\n");
cmCryptMechanism.mechanism = CKM_GOST28147;
cmCryptMechanism.pParameter = iv;
cmCryptMechanism.ulParameterLen = sizeof(iv);
printf("    mechanism type: CKM_GOST28147\n");
printf("    plain text:\n%s\n", pbPlainText);
print_hex(pbPlainText, strlen(pbPlainText));

rvResult = Pkcs11FuncList->C_EncryptInit(hSessionSend,
    &cmCryptMechanism,
    hKeyForWU);
printf("    encrypt initialization result: 0x%x\n", rvResult);

if (rvResult == CKR_OK)
{
    rvResult = Pkcs11FuncList->C_Encrypt(hSessionSend,
```

```
        pbPlainText ,
        ulPlainTextSize ,
        NULL,
        &ulCipherSize);

    if (rvResult == CKR_OK)
    {
        printf("    cipher size: %d\n", ulCipherSize);

        pbCipherText = (CK_BYTE_PTR)
            malloc(ulCipherSize * sizeof(CK_BYTE));
        rvResult = Pkcs11FuncList->C_Encrypt(hSessionSend ,
            pbPlainText ,
            ulPlainTextSize ,
            pbCipherText ,
            &ulCipherSize);
    }
}
else return rvResult;

printf("    encrypt result: 0x%x\n", rvResult);

if (rvResult != CKR_OK) return rvResult;

printf("    cipher text: \n");
print_hex(pbCipherText , ulCipherSize);

printf("Recipient decrypt with hUnwrappedRecpKey:\n");
rvResult = Pkcs11FuncList2->C_DecryptInit(hSessionRecp ,
    &cmCryptMechanism ,
    hUnwrappedRecpKey);
printf("    decrypt initialization result: 0x%x\n", rvResult);

if (rvResult == CKR_OK)
{
    rvResult = Pkcs11FuncList2->C_Decrypt(hSessionRecp ,
        pbCipherText ,
        ulCipherSize ,
        NULL,
        &ulDecryptedSize);

    if (rvResult == CKR_OK)
```

```
{
    printf("    decrypted text size: %d\n", ulDecryptedSize);

    pbDecryptedText = (CK_BYTE_PTR)
        malloc(ulDecryptedSize * sizeof(CK_BYTE));
    rvResult = Pkcs11FuncList2->C_Decrypt(hSessionRecp,
        pbCipherText,
        ulCipherSize,
        pbDecryptedText,
        &ulDecryptedSize);
}
}
else return rvResult;

printf("    decrypt result: 0x%x\n", rvResult);

if (rvResult != CKR_OK) return rvResult;

printf("    decrypted text: \n");
print_hex(pbDecryptedText, ulDecryptedSize);

if (ulDecryptedSize != ulPlainTextSize) {
    fprintf(stderr, "Invalid decrypted text size\n");
    return -1;
}

if (memcmp(pbDecryptedText, pbPlainText, ulPlainTextSize) !=
0) {
    fprintf(stderr, "Invalid decrypted text\n");
    return -1;
}
printf("CKM_GOST28147_KEY_WRAP test SUCCESS\n");

rvResult =
    Pkcs11FuncList->C_DestroyObject(hSessionSend, hKeyForWU);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult =
    Pkcs11FuncList->C_DestroyObject(hSessionSend, hSendPubKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult =
    Pkcs11FuncList->C_DestroyObject(hSessionSend, hSendPriKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult =
    Pkcs11FuncList2->C_DestroyObject(hSessionRecp, hRecpPubKey);
```

```

printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult =
    Pkcs11FuncList2->C_DestroyObject(hSessionRecv, hRecvPriKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult =
    Pkcs11FuncList->C_DestroyObject(hSessionSend, hSendDH_Key);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult =
    Pkcs11FuncList2->C_DestroyObject(hSessionRecv, hRecvDH_Key);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult =
    Pkcs11FuncList2->C_DestroyObject(hSessionRecv,
    hUnwrappedRecvKey);
printf("    C_DestroyObject result: 0x%x\n", rvResult);
rvResult = Pkcs11FuncList->C_CloseSession(hSessionSend);
printf("Sender close session result: 0x%x\n", rvResult);

rvResult = Pkcs11FuncList2->C_CloseSession(hSessionRecv);
printf("Recipient close session result: 0x%x\n", rvResult);

printf("SUCCESS\n");
return 0;
}

```

3.3.15 Шифрование ключей по ГОСТ Р34.10-2001

В данном примере демонстрируется шифрование секретного ключа механизмом СКМ_GOSTR3410_KEY_WRAP. Зашифрованный ключ представлен в виде транспортной DER-структуры, соответствующей ASN.1 типу GostR3410-KeyTransport, определенной в [9] п.4.2. Параметры шифрования передаются механизму в структуре СКМ_GOSTR3410_KEY_WRAP_PARAMS.

Листинг 3.24: ckm_gost3410_key_wrap.c

```

#include "test_common.h"

CK_RV test_crypto();
CK_RV test_gost3410_key_wrap(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {

```

```

    fprintf(stderr, "test_main failed: 0x%x\n", rc);
    return rc;
}
rc = test_crypto();
if (rc != CKR_OK) {
    fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
    return rc;
}
return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");

    rc = funcs->C_Login(hSession,
        CKU_USER, user_pin, strlen(user_pin));
    if (rc != CKR_OK) {
        fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
            rc);
        goto out_close;
    }
    fprintf(stderr, "C_Login success\n");

    rc = funcs->C_GetMechanismInfo(SlotId,
        CKM_GOSTR3410_KEY_WRAP, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n==== Mechanism CKM_GOSTR3410_KEY_WRAP "
            "not supported =====\n");
    } else {

```

```

    fprintf(stderr ,
"\n===== CKM_GOSTR3410_KEY_WRAP "
"test =====\n");
    rc = test_gost3410_key_wrap(hSession);
    if (rc != CKR_OK) {
        fprintf(stderr ,
            "ERROR CKM_GOSTR3410_KEY_WRAP failed, rc = 0x%x\n",
            rc);
    } else {
        fprintf(stderr , "CKM_GOSTR3410_KEY_WRAP test passed.\n");
    }
}

out_close:
    if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
        fprintf(stderr ,
            "Error: C_CloseSession failed with 0x%x\n", ret);
        rc = ret;
    }
    else {
        fprintf(stderr , "C_CloseSession success\n");
    }
}

out:
    return rc;
}

CK_RV test_gost3410_key_wrap(CK_SESSION_HANDLE sess)
{
    CK_RV rc;
    CK_BYTE value[1024];
    static CK_BYTE id[4];
    CK_ULONG len;
    CK_OBJECT_HANDLE send_pub_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE send_priv_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE recp_pub_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE recp_priv_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE cipher_key = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE unwrapped_cipher_key = CK_INVALID_HANDLE;
    CK_ATTRIBUTE attr;
    CK_MECHANISM mechanism_desc = {CKM_GOSTR3410_KEY_WRAP, NULL,
    0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;
    CK_MECHANISM mechanism_gen_desc =

```

```

{CKM_GOSTR3410_KEY_PAIR_GEN, NULL, 0};
CK_MECHANISM_PTR mechanism_gen = &mechanism_gen_desc;
CK_MECHANISM mechanism_gen_desc_512 =
{CKM_GOSTR3410_512_KEY_PAIR_GEN, NULL, 0};
CK_MECHANISM_PTR mechanism_gen_512 = &mechanism_gen_desc_512
;

static CK_BBOOL ltrue = CK_TRUE;
static CK_BBOOL lfalse = CK_FALSE;
static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
static CK_OBJECT_CLASS oclass_pub = CKO_PUBLIC_KEY;
static CK_OBJECT_CLASS oclass_priv = CKO_PRIVATE_KEY;
static CK_KEY_TYPE wrapping_key_type_256 = CKK_GOSTR3410;
static CK_KEY_TYPE wrapping_key_type_512 = CKK_GOSTR3410_512;
static CK_KEY_TYPE wrapped_key_type = CKK_GOST28147;
// PAR ECC A OID
static CK_BYTE gostR3410params_A [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01
};
// PAR ECC XA OID
static CK_BYTE gostR3410params_XA [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x24, 0x00
};
// PAR 512 A OID
static CK_BYTE gostR3410_2012_512_params [] = {
    0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x2, 0x01, 0x02, 0
    x01
};
// PAR HASH 1 OID
static CK_BYTE gostR3411params [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
};
// PAR 3411-2012-256 OID
static CK_BYTE gostR3411_2012_256_params [] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02
};
// PAR 3411-2012-512 OID
static CK_BYTE gostR3411_2012_512_params [] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03
};
// PAR CIPHER A OID
static CK_BYTE gost28147params_A [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
};
};

```

```

// PAR CIPHER B OID
static CK_BYTE gost28147params_B [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x02
};

// Cipher key value
static CK_BYTE cipher_key_val [] = {
    0xc3, 0x73, 0x0c, 0x5c, 0xbc, 0xca, 0xcf, 0x91,
    0x5a, 0xc2, 0x92, 0x67, 0x6f, 0x21, 0xe8, 0xbd,
    0x4e, 0xf7, 0x53, 0x31, 0xd9, 0x40, 0x5e, 0x5f,
    0x1a, 0x61, 0xdc, 0x31, 0x30, 0xa6, 0x50, 0x11
};

// Template for cipher key generation
static CK_ATTRIBUTE cipher_template [] = {
    { CKA_VALUE, cipher_key_val, sizeof(cipher_key_val) },
    { CKA_TOKEN, &lfalse, sizeof(lfalse) },
    { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) },
    { CKA_ENCRYPT, &ltrue, sizeof(ltrue) },
    { CKA_DECRYPT, &ltrue, sizeof(ltrue) },
    { CKA_WRAP, &ltrue, sizeof(ltrue) },
    { CKA_UNWRAP, &ltrue, sizeof(ltrue) },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_GOST28147PARAMS,
      gost28147params_A, sizeof(gost28147params_A) },
    { CKA_KEY_TYPE, &wrapped_key_type, sizeof(wrapped_key_type)
    }
};

static CK_BYTE ukm [] = {
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10,
};

// Вообще-то при упаковке значение УКМ должно быть равным NULL
,
// чтобы сгенерировалось случайное значение.
// Но мы здесь используем фиксированное значение, чтобы убедит
ься,
// что именно оно используется и попадает в результирующую стр
уктуру.
static CK_GOSTR3410_KEY_WRAP_PARAMS params = {
    gost28147params_A, sizeof(gost28147params_A),
    ukm, 8,
    CK_INVALID_HANDLE // Generate and use temporary ephemeral

```



```
key pair
};

// Key pair labels
static CK_BYTE priv_label [] = "Private Key Lissi 5312";
static CK_BYTE pub_label [] = "Public Key Lissi 5312";

// Templates for key pair generation.
// Template for token public key generation and search.
static CK_ATTRIBUTE pub_template_2001 [] = {
    { CKA_LABEL, pub_label, sizeof(pub_label) - 1 },
    { CKA_TOKEN, &ltrue, sizeof(ltrue) },
    { CKA_GOSTR3410PARAMS,
      gostR3410params_A, sizeof(gostR3410params_A) },
    { CKA_GOSTR3411PARAMS,
      gostR3411params, sizeof(gostR3411params) },
    { CKA_WRAP, &ltrue, sizeof(ltrue) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_CLASS, &oclass_pub, sizeof(oclass_pub) },
    { CKA_KEY_TYPE,
      &wrapping_key_type_256, sizeof(wrapping_key_type_256) },
};
static CK_ATTRIBUTE pub_template_2012_256 [] = {
    { CKA_LABEL, pub_label, sizeof(pub_label) - 1 },
    { CKA_TOKEN, &ltrue, sizeof(ltrue) },
    { CKA_GOSTR3410PARAMS,
      gostR3410params_A, sizeof(gostR3410params_A) },
    { CKA_GOSTR3411PARAMS,
      gostR3411_2012_256_params, sizeof(gostR3411_2012_256_params)
    },
    { CKA_WRAP, &ltrue, sizeof(ltrue) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_CLASS, &oclass_pub, sizeof(oclass_pub) },
    { CKA_KEY_TYPE,
      &wrapping_key_type_256, sizeof(wrapping_key_type_256) },
};
static CK_ATTRIBUTE pub_template_2012_512 [] = {
    { CKA_LABEL, pub_label, sizeof(pub_label) - 1 },
    { CKA_TOKEN, &ltrue, sizeof(ltrue) },
    { CKA_GOSTR3410PARAMS,
      gostR3410_2012_512_params, sizeof(gostR3410_2012_512_params)
    },
    { CKA_GOSTR3411PARAMS,
      gostR3411_2012_512_params, sizeof(gostR3411_2012_512_params)
    }
};
```

```

    },
    { CKA_WRAP, &ltrue, sizeof(ltrue) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_CLASS, &oclass_pub, sizeof(oclass_pub) },
    { CKA_KEY_TYPE,
      &wrapping_key_type_512, sizeof(wrapping_key_type_512) },
  };
  // Template for token private key generation and search.
  static CK_ATTRIBUTE priv_template_256[] = {
    { CKA_LABEL, priv_label, sizeof(priv_label) - 1 },
    { CKA_TOKEN, &ltrue, sizeof(ltrue) },
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) },
    { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
    { CKA_DERIVE, &ltrue, sizeof(ltrue) },
    { CKA_UNWRAP, &ltrue, sizeof(ltrue) },
    { CKA_DECRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_CLASS, &oclass_priv, sizeof(oclass_priv) },
    { CKA_KEY_TYPE,
      &wrapping_key_type_256, sizeof(wrapping_key_type_256) },
  };
  static CK_ATTRIBUTE priv_template_2012_512[] = {
    { CKA_LABEL, priv_label, sizeof(priv_label) - 1 },
    { CKA_TOKEN, &ltrue, sizeof(ltrue) },
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) },
    { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
    { CKA_DERIVE, &ltrue, sizeof(ltrue) },
    { CKA_UNWRAP, &ltrue, sizeof(ltrue) },
    { CKA_DECRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_CLASS, &oclass_priv, sizeof(oclass_priv) },
    { CKA_KEY_TYPE,
      &wrapping_key_type_512, sizeof(wrapping_key_type_512) },
  };
  // Buffer for public key value
  static CK_BYTE pub_value[128];

  // Template for public key object creation
  static CK_ATTRIBUTE new_pub_template[] = {
    { CKA_CLASS, &oclass_pub, sizeof(oclass_pub) },
    { CKA_KEY_TYPE,
      &wrapping_key_type_256, sizeof(wrapping_key_type_256) },
    { CKA_GOSTR3410PARAMS,
      gostR3410params_A, sizeof(gostR3410params_A) },
    { CKA_GOSTR3411PARAMS,
      gostR3411params, sizeof(gostR3411params) },
  };

```

```

    { CKA_WRAP, &ltrue, sizeof(CK_BBOOL) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VALUE, pub_value, 64 }
};
static CK_ATTRIBUTE new_pub_template_2012_256[] = {
    { CKA_CLASS, &oclass_pub, sizeof(oclass_pub) },
    { CKA_KEY_TYPE,
    &wrapping_key_type_256, sizeof(wrapping_key_type_256) },
    { CKA_GOSTR3410PARAMS,
    gostR3410params_A, sizeof(gostR3410params_A) },
    { CKA_GOSTR3411PARAMS,
    gostR3411_2012_256_params, sizeof(gostR3411_2012_256_params)
    },
    { CKA_WRAP, &ltrue, sizeof(CK_BBOOL) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VALUE, pub_value, 64 }
};
static CK_ATTRIBUTE new_pub_template_2012_512[] = {
    { CKA_CLASS, &oclass_pub, sizeof(oclass_pub) },
    { CKA_KEY_TYPE,
    &wrapping_key_type_512, sizeof(wrapping_key_type_512) },
    { CKA_GOSTR3410PARAMS,
    gostR3410_2012_512_params, sizeof(gostR3410_2012_512_params)
    },
    { CKA_GOSTR3411PARAMS,
    gostR3411_2012_512_params, sizeof(gostR3411_2012_512_params)
    },
    { CKA_WRAP, &ltrue, sizeof(CK_BBOOL) },
    { CKA_ENCRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VALUE, pub_value, 128 }
};
// Session public key object handle
CK_OBJECT_HANDLE new_pub_key = CK_INVALID_HANDLE;
/*
// Orlov KeyTransport
CK_BYTE orlov_data[] = {
    0x30, 0x81, 0xA4,
    0x30, 0x28,
    0x04, 0x20,
    0x50, 0x65, 0xC9, 0x40, 0x5D, 0x59, 0x65, 0x0A,
    0x08, 0xCD, 0xC2, 0x48, 0x60, 0xCB, 0x2D, 0x42,
    0xB3, 0xD7, 0x69, 0xEF, 0xF3, 0xCB, 0xA6, 0x88,
    0x55, 0x09, 0xF9, 0x48, 0xC6, 0x44, 0x9E, 0x5D,
    0x04, 0x04,

```

```

        0x2C, 0xC9, 0xAA, 0xEF,
    0xA0, 0x78,
        0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x1F, 0x01,
        0xA0, 0x63,
        0x30, 0x1C,
        0x06, 0x06, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x13,
        0x30, 0x12,
        0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x23, 0
x03,
        0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x1E, 0
x01,
        0x03, 0x43, 0x00,
        0x04, 0x40,
            0x70, 0xB2, 0x66, 0x34, 0x51, 0xCB, 0x88, 0x73,
            0x9E, 0xE9, 0x05, 0x6B, 0x80, 0x9C, 0xFE, 0x00,
            0xEC, 0x99, 0x7B, 0xB9, 0x8F, 0x5E, 0x30, 0x8B,
            0xA6, 0xCB, 0x5C, 0x67, 0x30, 0xA1, 0x31, 0x5A,
            0xB7, 0x92, 0x83, 0xA0, 0x27, 0x68, 0xFA, 0xAA,
            0x28, 0xD8, 0x90, 0x8E, 0xEB, 0x59, 0x2E, 0x50,
            0x7E, 0x42, 0x38, 0x2E, 0x5D, 0x15, 0xAA, 0xB9,
            0xB4, 0x1C, 0xE2, 0x81, 0x8C, 0x20, 0xB4, 0x6D,
        0x04, 0x08,
        0x80, 0x77, 0xF0, 0x58, 0x87, 0x03, 0x74, 0x03,
//        0x30, 0x36, 0x06, 0x09, 0x2A, // ???
};
// Orlov recipient private key
CK_BYTE orlov_recip_priv_key[] = {
    0x30, 0x45,
    0x02, 0x01, 0x00,
    0x30, 0x1C,
    0x06, 0x06, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x13,
    0x30, 0x12,
    0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x24, 0x00,
    0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x1E, 0x01,
    0x04, 0x22,
    0x04, 0x20,
    0x64, 0xA8, 0x2A, 0xAD, 0xC1, 0xAB, 0x9B, 0xB2,
    0xB1, 0x96, 0x99, 0xE9, 0x9E, 0xF6, 0x7D, 0xC6,
    0xDB, 0xCA, 0xB0, 0x36, 0x00, 0x3D, 0xD9, 0x77,
    0xCF, 0x96, 0xAF, 0x62, 0xB9, 0xE2, 0x3E, 0x07,
};
CK_BYTE orlov_priv_key_value[] = {
    0x64, 0xA8, 0x2A, 0xAD, 0xC1, 0xAB, 0x9B, 0xB2,
    0xB1, 0x96, 0x99, 0xE9, 0x9E, 0xF6, 0x7D, 0xC6,

```

```

    0xDB, 0xCA, 0xB0, 0x36, 0x00, 0x3D, 0xD9, 0x77,
    0xCF, 0x96, 0xAF, 0x62, 0xB9, 0xE2, 0x3E, 0x07,
};
*/
/*
// ТК 26 Закрытый ключ получателя
30 49
    02 01 00
30 1F
    06 08 2A 85 03 07 01 01 01 01
    30 13
        06 07 2A 85 03 02 02 24 00
        06 08 2A 85 03 07 01 01 02 02
04 23
    02 21
    00 ...
*/
CK_BYTE tc26_priv_key_value [] = {
    0x01, 0xB4, 0x2C, 0x61, 0xA1, 0xAE, 0xEF, 0x62,
    0xCE, 0xB9, 0x90, 0x3F, 0x8B, 0x56, 0x24, 0xB3,
    0x91, 0x9E, 0xEC, 0xEC, 0x43, 0x7F, 0x8A, 0x85,
    0x12, 0xCA, 0x7E, 0x43, 0xD0, 0x2E, 0x54, 0xE3,
};

// ТК 26 KeyTransport
CK_BYTE tc26_data [] = {
    0x30, 0x81, 0xA9,
    0x30, 0x28,
    0x04, 0x20,
    0x10, 0x8A, 0x8B, 0xF9, 0x80, 0x51, 0x42, 0xAE,
    0xF9, 0x1F, 0x0E, 0xD5, 0xAD, 0xE6, 0xDD, 0x3A,
    0x5B, 0x4E, 0x6C, 0x01, 0x4D, 0xBB, 0x93, 0x28,
    0xC0, 0x31, 0xE8, 0x06, 0x49, 0x76, 0xF2, 0x95,
    0x04, 0x04,
    0xAB, 0x49, 0x7F, 0x56,
    0xA0, 0x7D,
    0x06, 0x09, 0x2A, 0x85, 0x03, 0x07, 0x01, 0x02, 0x05, 0
x01, 0x01,
    0xA0, 0x66,
    0x30, 0x1F,
    0x06, 0x08, 0x2A, 0x85, 0x03, 0x07, 0x01, 0x01, 0x01
, 0x01,
    0x30, 0x13,
    0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x24, 0

```

```

x00 ,
    0x06, 0x08, 0x2A, 0x85, 0x03, 0x07, 0x01, 0x01, 0
x02, 0x02 ,
    0x03, 0x43, 0x00 ,
    0x04, 0x40 ,
    0xB4, 0x83, 0xB2, 0xBC, 0xC3, 0x57, 0x27, 0x2E ,
    0xAC, 0x57, 0x80, 0x0E, 0xC4, 0x9F, 0x5F, 0x9C ,
    0x1B, 0x2E, 0xB9, 0x7E, 0x78, 0x04, 0xB3, 0xEA ,
    0x34, 0xBC, 0x93, 0x57, 0x67, 0x4E, 0x43, 0x0C ,
    0xC0, 0xF7, 0x58, 0x3E, 0x4C, 0x34, 0xEC, 0x11 ,
    0xCB, 0xDE, 0xAA, 0x3A, 0xC8, 0xA7, 0xA0, 0xFF ,
    0xEF, 0x1E, 0x3A, 0x0B, 0xFE, 0xAE, 0xCA, 0xFD ,
    0xA7, 0x45, 0x4C, 0xED, 0x1A, 0xB0, 0x61, 0x3D ,
    0x04, 0x08 ,
    0x0D, 0x91, 0x85, 0x60, 0x71, 0xCF, 0x59, 0x8F ,
};
CK_ATTRIBUTE new_priv_template[] = {
    { CKA_CLASS, &oclass_priv, sizeof(oclass_priv) },
    { CKA_KEY_TYPE,
    &wrapping_key_type_256, sizeof(wrapping_key_type_256) },
    { CKA_GOSTR3410PARAMS,
    gostR3410params_XA, sizeof(gostR3410params_XA) },
    { CKA_GOSTR3411PARAMS,
    gostR3411_2012_256_params, sizeof(gostR3411_2012_256_params)
    },
    { CKA_SIGN, &ltrue, sizeof(CK_BBOOL) },
    { CKA_DECRYPT, &ltrue, sizeof(CK_BBOOL) },
    { CKA_DERIVE, &ltrue, sizeof(CK_BBOOL) },
    { CKA_VALUE, tc26_priv_key_value, sizeof(tc26_priv_key_value
    ) }
};
CK_OBJECT_HANDLE hObject = CK_INVALID_HANDLE;
CK_ULONG ulObjectCount = 0;
SYSTEMTIME t1, t2;
CK_ULONG diff;

// В примерах ТК 26 от КриптоКом при шифровании сообщений PKCS#7
// использована старая функция диверсификации VKO
// для новых алгоритмов 2012.
printf("===== TC 26 example =====\n"
);
// Распаковка данных ТК 26
rc = funcs->C_CreateObject(sess,
    new_priv_template,

```

```

    sizeof(new_priv_template)/sizeof(CK_ATTRIBUTE) ,
    &recp_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_CreateObject failed, rc = 0x%x\n" , __LINE__ , rc);
    return rc;
}
mechanism->pParameter = NULL;
mechanism->ulParameterLen = 0;

rc = funcs->C_UnwrapKey(sess , mechanism ,
    recp_priv_key , tc26_data , sizeof(tc26_data) , cipher_template
    + 1 ,
    sizeof(cipher_template)/sizeof(CK_ATTRIBUTE) - 1 ,
    &unwrapped_cipher_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_UnwrapKey failed, rc = 0x%x\n" , __LINE__ , rc);
    return rc;
}
printf("Key unwrapped OK\n");

printf("Cipher key to wrap:\n");
print_hex(cipher_key_val , 32);

printf("=====GOSTR3410-2001=====\n"
    " ");
// GOSTR3410-2001
// Generate sender key pair objects
rc = funcs->C_GenerateKeyPair(sess , mechanism_gen ,
    pub_template_2001 , sizeof(pub_template_2001)/sizeof(
    CK_ATTRIBUTE) ,
    priv_template_256 , sizeof(priv_template_256)/sizeof(
    CK_ATTRIBUTE) ,
    &send_pub_key , &send_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_GenerateKeyPair failed, rc = 0x%x\n" , __LINE__ , rc
    );
    return rc;
}
printf("Sender key pair generation OK\n");

// Generate recipient key pair objects

```

```
rc = funcs->C_GenerateKeyPair(sess, mechanism_gen,
    pub_template_2001, sizeof(pub_template_2001)/sizeof(
    CK_ATTRIBUTE),
    priv_template_256, sizeof(priv_template_256)/sizeof(
    CK_ATTRIBUTE),
    &recp_pub_key, &recp_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKeyPair failed, rc = 0x%x\n", __LINE__, rc
    );
    return rc;
}
printf("Recipient key pair generation OK\n");

// Get public key value
attr.type = CKA_VALUE;
attr.pValue = pub_value;
attr.ulValueLen = sizeof(pub_value);
rc = funcs->C_GetAttributeValue(sess, recp_pub_key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n", __LINE__,
    rc);
    return rc;
}
// gost_hexdump(stdout,
// "Recipient Public Key Value:", pub_value, attr.ulValueLen)
;

// Create session public key object from its value.
// There is no need to do it here but we make session public
// key for testing only.
rc = funcs->C_CreateObject(sess, new_pub_template,
    sizeof(new_pub_template)/sizeof(CK_ATTRIBUTE), &
    new_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_CreateObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

// Create cipher key object from its value
rc = funcs->C_CreateObject(sess, cipher_template,
    sizeof(cipher_template)/sizeof(CK_ATTRIBUTE), &cipher_key)
```



```
;
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_CreateObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

// Get cipher key to the value buffer
// from the cipher_key object (for testing).
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, cipher_key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n", __LINE__,
        rc);
    return rc;
}
len = attr.ulValueLen;
// gost_hexdump(stdout, "Cipher key value:", value, len);

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);

// avg_time = 0;
// max_time = 0;
// min_time = 0xFFFFFFFF;
// for (i=0; i < 10; i++) {
//     GetSystemTime(&t1);
params.pWrapOID = gost28147params_A;
params.ulWrapOIDLen = sizeof(gost28147params_A);
// Использовать закрытый ключ отправителя
// или CK_INVALID_HANDLE для генерации эфемерной ключевой пары
params.hKey = CK_INVALID_HANDLE; // send_priv_key;
// Get wrapping key length only.
len = sizeof(value);
rc = funcs->C_WrapKey(sess, mechanism,
    new_pub_key, cipher_key, NULL, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_WrapKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
```

```

}
// fprintf(stdout, "Wrapped key length = %d\n", len);

// Wrap cipher key with public key to the value buffer.
GetSystemTime(&t1);
rc = funcs->C_WrapKey(sess, mechanism,
new_pub_key, cipher_key, value, &len);
GetSystemTime(&t2);
diff = process_time(t1, t2);
fprintf(stderr, "C_WrapKey time: %ld msec\n", diff);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_WrapKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
printf("Key wrapped OK\n");
print_hex(value, len);

/*
// Без информации об открытом ключе должна получиться
// структура следующего вида:
0x30, 0x3f,
    0x30, 0x28,
        0x04, 0x20,
            0x76, 0x9e, 0xc1, 0x2e, 0x5c, 0x78, 0x66, 0x6a,
            0x9f, 0x56, 0x4c, 0xfe, 0x79, 0x65, 0x79, 0x6e,
            0x0e, 0x0e, 0xac, 0xd6, 0x36, 0x94, 0x9e, 0x39,
            0xf3, 0xc4, 0xc6, 0x97, 0xc9, 0x15, 0x9a, 0xe3,
        0x04, 0x04,
            0xe3, 0xd4, 0x3b, 0xf7,
    0xa0, 0x13,
        0x06, 0x07,
            0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
        0x04, 0x08,
            0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,

// При генерации эфемерной ключевой пары должна получиться
// структура следующего вида:
0x30, 0x81, 0xa4,
    0x30, 0x28,
        0x04, 0x20,
            0x65, 0x89, 0xd3, 0x41, 0x90, 0x9c, 0xb9, 0x03,
            0x45, 0x92, 0x13, 0x84, 0x57, 0x9b, 0xb8, 0x4e,
            0x82, 0x68, 0xb3, 0x7b, 0x80, 0x4f, 0x67, 0x15,

```

```

    0x84, 0xf5, 0x90, 0x5d, 0x0d, 0x58, 0x95, 0x61,
    0x04, 0x04,
    0x42, 0x84, 0x55, 0x80,
0xa0, 0x78,
    0x06, 0x07,
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
0xa0, 0x63,
    0x30, 0x1c,
    0x06, 0x06,
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x13,
    0x30, 0x12,
    0x06, 0x07,
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01,
    0x06, 0x07,
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01,
0x03, 0x43, 0x00,
    0x04, 0x40,
    0x0f, 0x1b, 0xd3, 0x67, 0x4d, 0xed, 0x83, 0x22,
    0x2a, 0xa0, 0x23, 0x4c, 0x55, 0x7a, 0x8a, 0xcd,
    0x71, 0x75, 0x11, 0x23, 0x67, 0x68, 0xcb, 0xfe,
    0xb8, 0x8f, 0x2d, 0xc5, 0x72, 0xb8, 0x3b, 0x6e,
    0xd2, 0x31, 0x38, 0xa6, 0xe4, 0xe3, 0x35, 0xbc,
    0x6d, 0x07, 0x1b, 0x57, 0xf8, 0xd7, 0x18, 0xa6,
    0x14, 0xf0, 0xe2, 0x3c, 0x65, 0x10, 0x5d, 0xa4,
    0x44, 0xaa, 0x1d, 0xea, 0x47, 0x81, 0x3e, 0x98,
0x04, 0x08,
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
*/

// Проверяем вариант с отсутствующим OID 3411 (0x05, 0x00)
// memcpy(value+0x4f+2, value+0x4f+9, len-(0x4f+9));
// len -= 7;

params.pWrapOID = NULL;
params.ulWrapOIDLen = 0;

// Использовать открытый ключ отправителя
// или CK_INVALID_HANDLE при генерации эфемерной ключевой пары
.
params.hKey = CK_INVALID_HANDLE; //send_pub_key;

GetSystemTime(&t1);
// Unwrap cipher key with recipient private key from the
value buffer.

```

```

// Don't use the first CKA_VALUE attribute -
// it will be added as a result of C_UnwrapKey for cipher_key.
rc = funcs->C_UnwrapKey(sess, mechanism,
    recp_priv_key, value, len, cipher_template + 1,
    sizeof(cipher_template)/sizeof(CK_ATTRIBUTE) - 1,
    &unwrapped_cipher_key);
GetSystemTime(&t2);
diff = process_time(t1, t2);
fprintf(stderr, "C_UnwrapKey time: %ld msec\n", diff);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_UnwrapKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
printf("Key unwrapped OK\n");

// Restore cipher key to the value buffer from the cipher_key
// object.
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, unwrapped_cipher_key, &
    attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n", __LINE__,
    rc);
    return rc;
}
len = attr.ulValueLen;
// Check cipher key value
if (len != sizeof(cipher_key_val)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
if (memcmp(value, cipher_key_val, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
printf("Unwrapped key is equal to source key\n");

rc = funcs->C_DestroyObject(sess, unwrapped_cipher_key);
if (rc != CKR_OK) {

```

```
fprintf(stderr ,
    "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
return rc;
}
//      GetSystemTime(&t2);
//      diff = process_time(t1, t2);
//      avg_time += diff;
//      if (diff < min_time)
//          min_time = diff;
//      if (diff > max_time)
//          max_time = diff;
//      }
//      printf("10 GOST R34.10-2001 Wrap/Unwrap operations: %ld \n
", avg_time );
//      printf("Minimum:                %ld \n", min_time )
;
//      printf("Maximum:                %ld \n", max_time )
;
//      printf("\n");

// Destroy session public key
rc = funcs->C_DestroyObject(sess , new_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
// Destroy cipher key
rc = funcs->C_DestroyObject(sess , cipher_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , recp_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , recp_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
```

```
    return rc;
}
rc = funcs->C_DestroyObject(sess, send_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess, send_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

printf("=====GOSTR3410-2012-256
=====\\n");
// GOSTR3410-2012-256
// Generate sender key pair objects
rc = funcs->C_GenerateKeyPair(sess, mechanism_gen,
    pub_template_2012_256,
    sizeof(pub_template_2012_256)/sizeof(CK_ATTRIBUTE),
    priv_template_256,
    sizeof(priv_template_256)/sizeof(CK_ATTRIBUTE),
    &send_pub_key, &send_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKeyPair failed, rc = 0x%x\n", __LINE__, rc
    );
    return rc;
}
printf("Sender key pair generation OK\\n");

// Generate recipient key pair objects
rc = funcs->C_GenerateKeyPair(sess, mechanism_gen,
    pub_template_2012_256,
    sizeof(pub_template_2012_256)/sizeof(CK_ATTRIBUTE),
    priv_template_256,
    sizeof(priv_template_256)/sizeof(CK_ATTRIBUTE),
    &recp_pub_key, &recp_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKeyPair failed, rc = 0x%x\n", __LINE__, rc
```

```
);
    return rc;
}
printf("Recipient key pair generation OK\n");

// Get public key value
attr.type = CKA_VALUE;
attr.pValue = pub_value;
attr.ulValueLen = sizeof(pub_value);
rc = funcs->C_GetAttributeValue(sess, recip_pub_key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n", __LINE__,
        rc);
    return rc;
}
// gost_hexdump(stdout,
// "Recipient Public Key Value:", pub_value, attr.ulValueLen)
;

// Create session public key object from its value.
// There is no need to do it here but we make session public
// key for testing only.
rc = funcs->C_CreateObject(sess, new_pub_template_2012_256,
    sizeof(new_pub_template_2012_256)/sizeof(CK_ATTRIBUTE),
    &new_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_CreateObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

// Create cipher key object from its value
rc = funcs->C_CreateObject(sess, cipher_template,
    sizeof(cipher_template)/sizeof(CK_ATTRIBUTE), &cipher_key)
;
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_CreateObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

// Get cipher key to the value buffer
// from the cipher_key object (for testing).
```

```

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, cipher_key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
len = attr.ulValueLen;
// gost_hexdump(stdout, "Cipher key value:", value, len);

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);

// avg_time = 0;
// max_time = 0;
// min_time = 0xFFFFFFFF;
// for (i=0; i < 10; i++) {
//     GetSystemTime(&t1);
params.pWrapOID = gost28147params_A;
params.ulWrapOIDLen = sizeof(gost28147params_A);
// Использовать закрытый ключ отправителя
// или CK_INVALID_HANDLE для генерации эфемерной ключевой пары
params.hKey = CK_INVALID_HANDLE; // send_priv_key;
// Get wrapping key length only.
len = sizeof(value);
rc = funcs->C_WrapKey(sess, mechanism,
    new_pub_key, cipher_key, NULL, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_WrapKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
// fprintf(stdout, "Wrapped key length = %d\n", len);

// Wrap cipher key with public key to the value buffer.
GetSystemTime(&t1);
rc = funcs->C_WrapKey(sess, mechanism,
    new_pub_key, cipher_key, value, &len);
GetSystemTime(&t2);
diff = process_time(t1, t2);

```



```

fprintf(stderr, "C_WrapKey time: %ld msec\n", diff );
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_WrapKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
printf("Key wrapped OK\n");
print_hex(value, len);

/*
// Без информации об открытом ключе должна получиться
// структура следующего вида:
0x30, 0x3f,
    0x30, 0x28,
        0x04, 0x20,
            0x76, 0x9e, 0xc1, 0x2e, 0x5c, 0x78, 0x66, 0x6a,
            0x9f, 0x56, 0x4c, 0xfe, 0x79, 0x65, 0x79, 0x6e,
            0x0e, 0x0e, 0xac, 0xd6, 0x36, 0x94, 0x9e, 0x39,
            0xf3, 0xc4, 0xc6, 0x97, 0xc9, 0x15, 0x9a, 0xe3,
        0x04, 0x04,
            0xe3, 0xd4, 0x3b, 0xf7,
    0xa0, 0x13,
        0x06, 0x07,
            0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
        0x04, 0x08,
            0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,

// При генерации эфемерной ключевой пары
// должна получиться структура следующего вида:
0x30, 0x81, 0xa7,
    0x30, 0x28,
        0x04, 0x20,
            0x78, 0x3e, 0xff, 0xba, 0x7c, 0xc9, 0x9d, 0xcb,
            0x45, 0x1f, 0x42, 0x76, 0xc9, 0x37, 0x47, 0xa2,
            0x54, 0x97, 0xaa, 0x49, 0x05, 0x74, 0x86, 0xe9,
            0xa9, 0x42, 0x88, 0xcb, 0x37, 0x67, 0x74, 0xfb,
        0x04, 0x04,
            0x9a, 0x7f, 0xcf, 0x45,
    0xa0, 0x7b,
        0x06, 0x07,
            0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
    0xa0, 0x66,
        0x30, 0x1f,
            0x06, 0x08,

```

```

    0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x01, 0x01,
    0x30, 0x13,
    0x06, 0x07,
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01,
    0x06, 0x08,
    0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02,
0x03, 0x43, 0x00,
    0x04, 0x40,
    0xb4, 0xd9, 0x79, 0x33, 0x59, 0x89, 0x1c, 0x4b,
    0x63, 0x2b, 0x96, 0x34, 0x69, 0x11, 0x87, 0x67,
    0x4f, 0xa4, 0xdf, 0xf1, 0x5a, 0x43, 0xf2, 0xb8,
    0x22, 0xd2, 0x96, 0x1b, 0xa5, 0x48, 0xda, 0xff,
    0x31, 0x9e, 0xf5, 0x59, 0xc6, 0xdc, 0x4b, 0x4a,
    0xe6, 0x0b, 0x37, 0x63, 0xbc, 0xd3, 0x7f, 0x58,
    0x12, 0xed, 0x19, 0x88, 0x49, 0xb6, 0x07, 0x08,
    0x57, 0x9f, 0xa8, 0x2f, 0x1d, 0xb1, 0x05, 0x68,
0x04, 0x08,
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
*/

// Проверяем вариант с отсутствующим OID 3411 (0x05, 0x00)
// memcpy(value+0x4f+2, value+0x4f+9, len-(0x4f+9));
// len -= 7;

params.pWrapOID = NULL;
params.ulWrapOIDLen = 0;

// Использовать открытый ключ отправителя
// или CK_INVALID_HANDLE при генерации эфемерной ключевой пары
params.hKey = CK_INVALID_HANDLE; // send_pub_key;

GetSystemTime(&t1);
// Unwrap cipher key with recipient private key from the
value buffer.
// Don't use the first CKA_VALUE attribute -
// it will be added as a result of C_UnwrapKey for cipher_key.
rc = funcs->C_UnwrapKey(sess, mechanism,
    recp_priv_key, value, len, cipher_template + 1,
    sizeof(cipher_template)/sizeof(CK_ATTRIBUTE) - 1,
    &unwrapped_cipher_key);
GetSystemTime(&t2);
diff = process_time(t1, t2);
fprintf(stderr, "C_UnwrapKey time: %ld msec\n", diff);

```

```
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_UnwrapKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
printf("Key unwrapped OK\n");

// Restore cipher key to the value buffer from the cipher_key
// object.
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, unwrapped_cipher_key,
    &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
len = attr.ulValueLen;
// Check cipher key value
if (len != sizeof(cipher_key_val)) {
    fprintf(stderr,
        "%4d: Invalid result length: %d\n", __LINE__, len);
    return -1;
}
if (memcmp(value, cipher_key_val, len) != 0) {
    fprintf(stderr, "%4d: Invalid result value\n", __LINE__);
    return -2;
}
printf("Unwrapped key is equal to source key\n");
rc = funcs->C_DestroyObject(sess, unwrapped_cipher_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
//     GetSystemTime(&t2);
//     diff = process_time(t1, t2);
//     avg_time += diff;
//     if (diff < min_time)
//         min_time = diff;
```

```
//      if (diff > max_time)
//          max_time = diff;
//      }
//      printf("10 GOST R34.10-2001 Wrap/Unwrap operations:  %ld \n
//      ", avg_time );
//      printf("Minimum:                                %ld \n", min_time )
//      ;
//      printf("Maximum:                                %ld \n", max_time )
//      ;
//      printf("\n");

// Destroy session public key
rc = funcs->C_DestroyObject(sess , new_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
// Destroy cipher key
rc = funcs->C_DestroyObject(sess , cipher_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , recp_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , recp_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , send_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
}
```

```
rc = funcs->C_DestroyObject(sess, send_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

printf("=====GOSTR3410-2012-512
=====\n");
// GOSTR3410-2012-512
// Generate sender key pair objects
rc = funcs->C_GenerateKeyPair(sess, mechanism_gen_512,
    pub_template_2012_512,
    sizeof(pub_template_2012_512)/sizeof(CK_ATTRIBUTE),
    priv_template_2012_512,
    sizeof(priv_template_2012_512)/sizeof(CK_ATTRIBUTE),
    &send_pub_key, &send_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKeyPair failed, rc = 0x%x\n", __LINE__, rc
    );
    return rc;
}
printf("Sender key pair generation OK\n");

// Generate recipient key pair objects
rc = funcs->C_GenerateKeyPair(sess, mechanism_gen_512,
    pub_template_2012_512,
    sizeof(pub_template_2012_512)/sizeof(CK_ATTRIBUTE),
    priv_template_2012_512,
    sizeof(priv_template_2012_512)/sizeof(CK_ATTRIBUTE),
    &recp_pub_key, &recp_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKeyPair failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
printf("Recipient key pair generation OK\n");

// Get public key value
attr.type = CKA_VALUE;
attr.pValue = pub_value;
attr.ulValueLen = sizeof(pub_value);
```

```
rc = funcs->C_GetAttributeValue(sess, recip_pub_key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
// gost_hexdump(stdout,
// "Recipient Public Key Value:", pub_value, attr.ulValueLen)
;

// Create session public key object from its value.
// There is no need to do it here but we make session public
// key for testing only.
rc = funcs->C_CreateObject(sess, new_pub_template_2012_512,
    sizeof(new_pub_template_2012_512)/sizeof(CK_ATTRIBUTE),
    &new_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_CreateObject failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

// Create cipher key object from its value
rc = funcs->C_CreateObject(sess, cipher_template,
    sizeof(cipher_template)/sizeof(CK_ATTRIBUTE), &cipher_key)
;
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_CreateObject failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

// Get cipher key to the value buffer
// from the cipher_key object (for testing).
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, cipher_key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
```

```

    __LINE__, rc );
    return rc;
}
len = attr.ulValueLen;
// gost_hexdump(stdout, "Cipher key value:", value, len);

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);

// avg_time = 0;
// max_time = 0;
// min_time = 0xFFFFFFFF;
// for (i=0; i < 10; i++) {
//     GetSystemTime(&t1);
params.pWrapOID = gost28147params_A;
params.ulWrapOIDLen = sizeof(gost28147params_A);
// Использовать закрытый ключ отправителя
// или CK_INVALID_HANDLE для генерации эфемерной ключевой пары
.
params.hKey = CK_INVALID_HANDLE; // send_priv_key;
params.ulUKMLen = 16;
// Get wrapping key length only.
len = sizeof(value);
rc = funcs->C_WrapKey(sess, mechanism,
    new_pub_key, cipher_key, NULL, &len);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_WrapKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
// fprintf(stdout, "Wrapped key length = %d\n", len);

// Wrap cipher key with public key to the value buffer.
GetSystemTime(&t1);
rc = funcs->C_WrapKey(sess, mechanism,
    new_pub_key, cipher_key, value, &len);
GetSystemTime(&t2);
diff = process_time(t1, t2);
fprintf(stderr, "C_WrapKey time: %ld msec\n", diff );
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_WrapKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
}

```

```
printf("Key wrapped OK\n");
print_hex(value, len);

/*
// При отсутствии информации об открытом ключе отправителя
// должна получиться структура следующего вида:
0x30, 0x47,
// Зашифрованный ключ
0x30, 0x28,
0x04, 0x20,
0xc7, 0xf7, 0x4a, 0x6d, 0xdc, 0x98, 0x20, 0x60,
0x0e, 0xb9, 0x0d, 0xfa, 0x2e, 0x85, 0x76, 0xec,
0xe4, 0x2b, 0x18, 0x36, 0x95, 0x38, 0xa7, 0x74,
0x62, 0xf5, 0x23, 0x9b, 0x2d, 0x0d, 0xf4, 0x10,
0x04, 0x04,
0x5f, 0xf4, 0x84, 0x38,
// Параметры шифрования
0xa0, 0x1b,
// Параметры алгоритма шифрования
0x06, 0x07,
0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
// UKM
0x04, 0x10,
0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10,
// Единственное отличие варианта 512 от варианта 256 - это 16
байтов UKM.

// При генерации эфемерной ключевой пары
// должна получиться структура следующего вида:
0x30, 0x81, 0xf5,
// Зашифрованный ключ
0x30, 0x28,
0x04, 0x20,
0xc3, 0xa3, 0x8f, 0x1f, 0x52, 0x03, 0xce, 0xcd,
0xc7, 0x37, 0xde, 0xb1, 0xf3, 0xd8, 0xed, 0xe4,
0x3e, 0x79, 0xd4, 0xeb, 0x28, 0x2f, 0xef, 0x99,
0x95, 0x5e, 0xb2, 0xf9, 0x9f, 0x8a, 0x9a, 0xdc,
0x04, 0x04,
0xb5, 0xd2, 0x31, 0x0e,
// Параметры шифрования
0xa0, 0x81, 0xc8,
// Параметры алгоритма шифрования
0x06, 0x07,
```



```
    0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01,
// Информация об открытом ключе отправителя
0xa0, 0x81, 0xaa,
    0x30, 0x21,
    0x06, 0x08,
    0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x01, 0x02,
    0x30, 0x15,
    0x06, 0x09,
    0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01, 0x02, 0
x01,
    0x06, 0x08,
    0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03,
0x03, 0x81, 0x84, 0x00,
    0x04, 0x81, 0x80,
    0x76, 0x88, 0x5d, 0x7e, 0x6c, 0xa1, 0x0a, 0x3b,
    0xa0, 0x3a, 0x1b, 0xd8, 0x44, 0x6d, 0xad, 0x3a,
    0x8a, 0xec, 0x3d, 0x0d, 0xd1, 0x7f, 0x78, 0xd7,
    0x02, 0xdc, 0x57, 0x37, 0xc0, 0x9e, 0xc0, 0xeb,
    0x2b, 0x68, 0xc5, 0x73, 0x5b, 0x97, 0x91, 0x46,
    0x7b, 0x30, 0x47, 0x8f, 0x3e, 0x76, 0x45, 0x11,
    0xe0, 0x97, 0x83, 0xa0, 0x34, 0xc5, 0x38, 0xee,
    0x95, 0x8b, 0x8b, 0xce, 0xf4, 0xb1, 0xad, 0x3d,
    0x8a, 0x5a, 0x2f, 0x5c, 0x93, 0x10, 0xb5, 0x2d,
    0x6b, 0x91, 0x82, 0xf7, 0x0f, 0x4c, 0xd7, 0x62,
    0x8f, 0x58, 0x2a, 0x2a, 0x9e, 0x2b, 0x9b, 0x96,
    0xce, 0x89, 0x66, 0x49, 0xc0, 0xc8, 0x69, 0x3b,
    0x1d, 0xdc, 0x6a, 0xba, 0x3c, 0x4a, 0xdf, 0x2e,
    0xad, 0x16, 0x96, 0x62, 0x42, 0xe4, 0x5d, 0xa9,
    0xbb, 0x85, 0x30, 0x37, 0xb0, 0xe0, 0xca, 0xc5,
    0x2f, 0x30, 0x12, 0x58, 0x6c, 0x0e, 0xd5, 0x3b,
// UKM
0x04, 0x10,
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f, 0x10,
*/

// Проверяем вариант с отсутствующим OID 3411 (0x05, 0x00)
// memcpu(value+0x4f+2, value+0x4f+9, len-(0x4f+9));
// len -= 7;

params.pWrapOID = NULL;
params.ulWrapOIDLen = 0;

// Использовать открытый ключ отправителя
```

```

// или CK_INVALID_HANDLE при генерации эфемерной ключевой пары
.
params.hKey = CK_INVALID_HANDLE;//send_pub_key;

    GetSystemTime(&t1);
    // Unwrap cipher key with recipient private key from the
    value buffer.
    // Don't use the first CKA_VALUE attribute -
    // it will be added as a result of C_UnwrapKey for cipher_key.
    rc = funcs->C_UnwrapKey(sess, mechanism,
        recp_priv_key, value, len, cipher_template + 1,
        sizeof(cipher_template)/sizeof(CK_ATTRIBUTE) - 1,
        &unwrapped_cipher_key);
    GetSystemTime(&t2);
    diff = process_time(t1, t2);
    fprintf(stderr, "C_UnwrapKey time: %ld msec\n", diff );
    if (rc != CKR_OK) {
        fprintf(stderr,
            "%4d: C_UnwrapKey failed, rc = 0x%x\n",
            __LINE__, rc);
        return rc;
    }
    printf("Key unwrapped OK\n");

// Restore cipher key to the value buffer from the cipher_key
object.
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, unwrapped_cipher_key,
    &attr, 1);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
            __LINE__, rc);
        return rc;
    }
    len = attr.ulValueLen;
    // Check cipher key value
    if (len != sizeof(cipher_key_val)) {
        fprintf(stderr,
            "%4d: Invalid result length: %d\n", __LINE__, len);
        return -1;
    }
}

```

```
if (memcmp(value , cipher_key_val , len) != 0) {
    fprintf(stderr , "%4d: Invalid result value\n" , __LINE__);
    return -2;
}
printf("Unwrapped key is equal to source key\n");

rc = funcs->C_DestroyObject(sess , unwrapped_cipher_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n" ,
        __LINE__ , rc);
    return rc;
}
//      GetSystemTime(&t2);
//      diff = process_time(t1 , t2);
//      avg_time += diff;
//      if (diff < min_time)
//          min_time = diff;
//      if (diff > max_time)
//          max_time = diff;
//  }
//  printf("10 GOST R34.10-2001 Wrap/Unwrap operations:  %ld \n
" , avg_time );
//  printf("Minimum:                %ld \n" , min_time )
;
//  printf("Maximum:                %ld \n" , max_time )
;
//  printf("\n");

// Destroy session public key
rc = funcs->C_DestroyObject(sess , new_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n" , __LINE__ , rc);
    return rc;
}
// Destroy cipher key
rc = funcs->C_DestroyObject(sess , cipher_key);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "%4d: C_DestroyObject failed, rc = 0x%x\n" , __LINE__ , rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , recp_pub_key);
```

```
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess, recip_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess, send_pub_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess, send_priv_key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

printf("SUCCESS\n");
rc = CKR_OK;

return rc;
}
```

3.3.16 Генерация ключа шифрования на пароле

При работе с транспортным контейнером типа PKCS#12 требуется генерировать ключ шифрования на пароле. В данном примере это делается с помощью дополнительного механизма CKM_PKCS5_PBKD2, в соответствии с рекомендациями Рабочей группы ТК 26.

Листинг 3.25: ckm_pkcs5_pbkd2.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_pkcs5_pbkd2_key_gen(CK_SESSION_HANDLE sess);
```

```

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
    /*
    // log in as normal user
    rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
        user_pin));
    if (rc != CKR_OK) {
        fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
            rc);
        goto out_close;
    }
    fprintf(stderr, "C_Login success\n");
    */
    rc = funcs->C_GetMechanismInfo(SlotId, CKM_PKCS5_PBKD2, &minfo
    );
}

```

```

    if (rc != CKR_OK) {
        fprintf(stderr,
"\n===== Mechanism CKM_PKCS5_PBKD2 not supported =====\n"
        n");
    } else {
        fprintf(stderr,
"\n===== CKM_PKCS5_PBKD2 test =====\n"
        n");
        rc = test_pkcs5_pbkd2_key_gen(hSession);
        if (rc != CKR_OK) {
            fprintf(stderr,
                "ERROR CKM_PKCS5_PBKD2 failed, rc = 0x%x\n", rc);
        } else {
            fprintf(stderr, "CKM_PKCS5_PBKD2 test passed.\n");
        }
    }

    if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
        fprintf(stderr,
            "Error: C_CloseSession failed with 0x%x\n", ret);
        rc = ret;
    }
    else {
        fprintf(stderr, "C_CloseSession success\n");
    }
}

out:
    return rc;
}

CK_RV test_pkcs5_pbkd2_key_gen(CK_SESSION_HANDLE sess)
{
    CK_RV rc = CKR_OK;
    CK_BYTE value[256];
    CK_ULONG len;
    CK_MECHANISM mechanism_desc = {CKM_PKCS5_PBKD2, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    CK_OBJECT_HANDLE key = CK_INVALID_HANDLE;
    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    static CK_BYTE gost28147_A [] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
    };
};

```

```
CK_OBJECT_CLASS lclass = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_GOST28147;
CK_CHAR label [] = "A GOST28147 secret key object";
CK_PKCS5_PBKD2_PARAMS params;
// CryptoPro gostR3411 HASH1 Param Set
static CK_BYTE gost_3411_94_par_oid [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
};
static CK_BYTE gost_3411_2012_256_oid [] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02
};
static CK_BYTE gost_3411_2012_512_oid [] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03
};
CK_ATTRIBUTE attr;

static CK_BYTE salt1 [] =
{ 0x12, 0x34, 0x56, 0x78, 0x78, 0x56, 0x34 };
static CK_ULONG iter1 = 5;
static CK_UTF8CHAR_PTR password1 = "password+";
static CK_ULONG password1_len = 9;
static CK_BYTE et1 [] = {
    0x35, 0x0e, 0x07, 0xc5, 0xb7, 0xfc, 0xe8, 0xe2,
    0xa0, 0xa8, 0xdf, 0xc3, 0x92, 0xc0, 0x49, 0x96,
    0x4c, 0xb1, 0x66, 0x31, 0x9b, 0x58, 0x6e, 0x9f,
    0x46, 0x22, 0x86, 0x76, 0x5d, 0x63, 0xdb, 0xf3,
};
static CK_BYTE salt2 [] = {
    0xbf, 0xb2, 0xd3, 0x23, 0x3e, 0xe6, 0x21, 0xd8
};
static CK_ULONG iter2 = 2048;
static CK_UTF8CHAR_PTR password2 = "1111";
static CK_ULONG password2_len = 4;
static CK_BYTE et2 [] = {
    0x76, 0xf1, 0xf3, 0x45, 0x6b, 0x43, 0x48, 0x99,
    0xe0, 0xfc, 0xda, 0xf8, 0x49, 0xc2, 0x68, 0x46,
    0x7b, 0xd5, 0x34, 0x5b, 0x25, 0x09, 0x74, 0x4e,
    0x0b, 0x32, 0x7b, 0xb9, 0x1e, 0x8f, 0xfa, 0x1a,
};
static CK_BYTE salt3 [] = {
    0x8C, 0x5B, 0x3E, 0xE5, 0xCD, 0x27, 0xFA, 0x28
};
static CK_ULONG iter3 = 2048;
static CK_UTF8CHAR_PTR password3 = "4444";
```

```

static CK_ULONG password3_len = 4;
static CK_BYTE et3 [] = {
    0xDB, 0xD1, 0x28, 0x79, 0xF6, 0xFC, 0x2C, 0x0C,
    0x8D, 0xC8, 0x06, 0x18, 0x15, 0xDB, 0xF9, 0x8B,
    0x58, 0xA8, 0x9C, 0x3F, 0x55, 0x96, 0x13, 0xDC,
    0xDB, 0x18, 0xD9, 0x0A, 0x84, 0xF2, 0x53, 0x8E,
};
static CK_BYTE salt4 [] = "salt";
static CK_ULONG iter4 = 1;
static CK_UTF8CHAR_PTR password4 = "password";
static CK_ULONG password4_len = 8;
static CK_BYTE et4 [] = {
    0x64, 0x77, 0x0a, 0xf7, 0xf7, 0x48, 0xc3, 0xb1,
    0xc9, 0xac, 0x83, 0x1d, 0xbc, 0xfd, 0x85, 0xc2,
    0x61, 0x11, 0xb3, 0x0a, 0x8a, 0x65, 0x7d, 0xdc,
    0x30, 0x56, 0xb8, 0x0c, 0xa7, 0x3e, 0x04, 0x0d,
};
static CK_ULONG iter5 = 4096;
static CK_BYTE et5 [] = {
    0xe5, 0x2d, 0xeb, 0x9a, 0x2d, 0x2a, 0xaf, 0xf4,
    0xe2, 0xac, 0x9d, 0x47, 0xa4, 0x1f, 0x34, 0xc2,
    0x03, 0x76, 0x59, 0x1c, 0x67, 0x80, 0x7f, 0x04,
    0x77, 0xe3, 0x25, 0x49, 0xdc, 0x34, 0x1b, 0xc7,
};

CK_ATTRIBUTE key_template [] = {
    { CKA_CLASS, &lclass, sizeof(lclass) },
    { CKA_KEY_TYPE, &keyType, sizeof(keyType) },
    { CKA_TOKEN, &lfalse, sizeof(lfalse) },
    { CKA_LABEL, label, sizeof(label) },
    { CKA_ENCRYPT, &ltrue, sizeof(ltrue) },
    { CKA_GOST28147_PARAMS,
      gost28147_A, sizeof(gost28147_A) },
    { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) }
};
/*
typedef struct CK_PKCS5_PBKD2_PARAMS {
    CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE      saltSource;
    CK_VOID_PTR
    pSaltSourceData;
    CK_ULONG
    ulSaltSourceDataLen;
    CK_ULONG                                iterations;
}

```



```

        CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf;
        CK_VOID_PTR                                pPrfData;
        CK_ULONG                                   ulPrfDataLen;
        CK_UTF8CHAR_PTR                           pPassword;
        CK_ULONG_PTR                               ulPasswordLen
    };
} CK_PKCS5_PBKD2_PARAMS;
*/

params.saltSource = CKZ_SALT_SPECIFIED;
params.pSaltSourceData = salt1;
params.ulSaltSourceDataLen = sizeof(salt1);
params.iterations = iter1;
params.prf = CKP_PKCS5_PBKD2_HMAC_GOSTR3411;
    params.pPrfData = gost_3411_94_par_oid;
    params.ulPrfDataLen = sizeof(gost_3411_94_par_oid);
params.pPassword = password1;
params.ulPasswordLen = &password1_len;

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);
rc = funcs->C_GenerateKey(sess, mechanism,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKey failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
len = attr.ulValueLen;

rc = funcs->C_DestroyObject(sess, key);

```

```
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

CHECK(value, len, et1);

params.saltSource = CKZ_SALT_SPECIFIED;
params.pSaltSourceData = salt2;
params.ulSaltSourceDataLen = sizeof(salt2);
params.iterations = iter2;
params.prf = CKP_PKCS5_PBKD2_HMAC_GOSTR3411;
    params.pPrfData = gost_3411_94_par_oid;
    params.ulPrfDataLen = sizeof(gost_3411_94_par_oid);
params.pPassword = password2;
params.ulPasswordLen = &password2_len;

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);
rc = funcs->C_GenerateKey(sess, mechanism,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKey failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
len = attr.ulValueLen;

rc = funcs->C_DestroyObject(sess, key);
```

```
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

CHECK(value, len, et2);

params.saltSource = CKZ_SALT_SPECIFIED;
params.pSaltSourceData = salt3;
params.ulSaltSourceDataLen = sizeof(salt3);
params.iterations = iter3;
params.prf = CKP_PKCS5_PBKD2_HMAC_GOSTR3411;
    params.pPrfData = gost_3411_94_par_oid;
    params.ulPrfDataLen = sizeof(gost_3411_94_par_oid);
params.pPassword = password3;
params.ulPasswordLen = &password3_len;

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);
rc = funcs->C_GenerateKey(sess, mechanism,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKey failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
len = attr.ulValueLen;

rc = funcs->C_DestroyObject(sess, key);
```

```
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

CHECK(value, len, et3);

params.saltSource = CKZ_SALT_SPECIFIED;
params.pSaltSourceData = salt4;
params.ulSaltSourceDataLen = strlen(salt4);
params.iterations = iter4;
params.prf = CKP_PKCS5_PBKD2_HMAC_GOSTR3411;
    params.pPrfData = gost_3411_2012_256_oid;
    params.ulPrfDataLen = sizeof(gost_3411_2012_256_oid);
params.pPassword = password4;
params.ulPasswordLen = &password4_len;

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);
rc = funcs->C_GenerateKey(sess, mechanism,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKey failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}
len = attr.ulValueLen;

rc = funcs->C_DestroyObject(sess, key);
```

```
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n",
        __LINE__, rc);
    return rc;
}

CHECK(value, len, et4);
/*
// Выполняется примерно 13 секунд.
params.iterations = iter5;

mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);
rc = funcs->C_GenerateKey(sess, mechanism,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE), &
    key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GenerateKey failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_GetAttributeValue failed, rc = 0x%x\n", __LINE__,
        rc);
    return rc;
}
len = attr.ulValueLen;

rc = funcs->C_DestroyObject(sess, key);
if (rc != CKR_OK) {
    fprintf(stderr,
        "%4d: C_DestroyObject failed, rc = 0x%x\n", __LINE__, rc);
    return rc;
}

CHECK(value, len, et5);
*/
```

```
printf("SUCCESS\n");
rc = CKR_OK;

return rc;
}
```

3.3.17 Генерация ключа аутентификации на пароле

Дополнительный механизм CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC вырабатывает на пароле так называемый ключ аутентификации, значение которого используется для проверки целостности транспортного контейнера PKCS#12. Заметим, что такой ключ имеет тип CKK_GENERIC_SECRET.

Листинг 3.26: ckm_pba_gostr3411_with_gostr3411_hmac.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_pba_gost3411_with_gost3411_hmac(CK_SESSION_HANDLE
    sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
```

```

    CKF_RW_SESSION | CKF_SERIAL_SESSION,
    NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
/*
// log in as normal user
rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
    user_pin));
    if (rc != CKR_OK) {
        fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
            rc);
        goto out_close;
    }
    fprintf(stderr, "C_Login success\n");
*/
    rc = funcs->C_GetMechanismInfo(SlotId,
        CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n===== Mechanism CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC "
            "not supported =====\n");
    } else {
        fprintf(stderr,
            "\n===== CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC "
            "test =====\n");
        rc = test_pba_gost3411_with_gost3411_hmac(hSession);
        if (rc != CKR_OK) {
            fprintf(stderr,
                "ERROR CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC failed, "
                "rc = 0x%x\n",
                rc);
        } else {
            fprintf(stderr,
                "CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC test passed.\n");
        }
    }
}

    if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
        fprintf(stderr, "Error: C_CloseSession failed with 0x%x\n",
            ret);
    }
}

```

```

    rc = ret;
}
else {
    fprintf(stderr, "C_CloseSession success\n");
}

out:
    return rc;
}

CK_RV test_pba_gost3411_with_gost3411_hmac(CK_SESSION_HANDLE
    sess)
{
    int rc = 0;
    CK_BYTE value[256];
    CK_ULONG len;
    CK_MECHANISM mechanism_desc =
        {CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;
    CK_OBJECT_HANDLE key = CK_INVALID_HANDLE;
    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    CK_GOSTR3411_PBE_PARAMS params;
    CK_ATTRIBUTE attr;
    CK_KEY_TYPE key_type = 0;
    // CryptoPro gostR3411 A Param Set
    static CK_BYTE oid_default[] = {
        0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01,
    };
    // Real PKCS#12 params and ethalon
    static CK_BYTE salt4[] = {
        0xDF, 0x7C, 0x5C, 0x10, 0xA4, 0xD5, 0x22, 0x62,
    };
    static CK_ULONG iter4 = 2048;
    // Use UTF-8 password here
    // Converting from UTF-8 to UTF-16 in soft token
    static CK_UTF8CHAR password4[] = {0x34, 0x34, 0x34, 0x34};
    static CK_ULONG password4_len = 4;
    static CK_BYTE et14[] = {
        0x93, 0x88, 0x91, 0x11, 0x20, 0x43, 0xC4, 0xD1,
        0xCA, 0x23, 0x82, 0xEF, 0x86, 0x4A, 0x67, 0x31,
        0xBD, 0x29, 0xEF, 0x82, 0x94, 0xDD, 0x23, 0x50,
        0x63, 0x0B, 0xCB, 0x1E, 0x5A, 0xC9, 0x06, 0xC3,
    };
};

```



```

static CK_ATTRIBUTE key_template [] = {
    { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) }
};

params.pOID = oid_default;
params.ulOIDLen = sizeof(oid_default);
params.pSalt = salt4;
params.ulSaltLen = sizeof(salt4);
params.ulIteration = iter4;
params.pPassword = password4;
params.ulPasswordLen = password4_len;
mechanism->pParameter = &params;
mechanism->ulParameterLen = sizeof(params);
// Механизм СКМ_PBA_GOSTR3411_WITH_GOSTR3411_HMAC вырабатывает
// на пароле так называемый ключ аутентификации,
// значение которого используется для проверки целостности
// транспортного контейнера PKCS#12.
rc = funcs->C_GenerateKey(sess, mechanism,
    key_template,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE), &key);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GenerateKey failed: 0x%x\n", rc);
    goto end;
}
attr.type = CKA_KEY_TYPE;
attr.pValue = &key_type;
attr.ulValueLen = sizeof(key_type);
rc = funcs->C_GetAttributeValue(sess, key, &attr, 1);
len = attr.ulValueLen;
if (key_type != CKK_GENERIC_SECRET) {
    fprintf(stderr,
        "Key type 0x%x != CKK_GENERIC_SECRET\n", key_type);
    rc = -1;
    goto end;
}
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, key, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    goto end;
}
}

```

```

len = attr.ulValueLen;

printf("Generated PBA key value:\n");
print_hex(value, len);

CHECK(value, len, et14);

printf("SUCCESS\n");
rc = CKR_OK;
end:
if (key != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess, key);
}
return rc;
}

```

3.3.18 Механизмы для TLS

В LS11SW поддерживаются дополнительные механизмы для поддержки протокола TLS:

CKM_TLS_GOST_PRF,
 CKM_TLS_GOST_PRE_MASTER_KEY_GEN,
 CKM_TLS_GOST_MASTER_KEY_DERIVE и
 CKM_TLS_GOST_KEY_AND_MAC_DERIVE.

Ниже приводятся примеры, демонстрирующие применение этих механизмов.

CKM_TLS_GOST_PRF

Листинг 3.27: ckm_tls_gost_prf.c

```

#include "test_common.h"

CK_RV test_crypto();
CK_RV test_tls_prf(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
}

```

```

    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
    /*
    // log in as normal user
    rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
        user_pin));
    if (rc != CKR_OK) {
        fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
            rc);
        goto out_close;
    }
    fprintf(stderr, "C_Login success\n");
    */

    rc = funcs->C_GetMechanismInfo(SlotId, CKM_TLS_GOST_PRF, &
        minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n===== Mechanism CKM_TLS_GOST_PRF not supported
            =====\n");
    } else {
        fprintf(stderr,
            "\n===== CKM_TLS_GOST_PRF test
            =====\n");
    }
}

```

```

rc = test_tls_prf(hSession);
if (rc != CKR_OK) {
    fprintf(stderr,
            "ERROR CKM_TLS_GOST_PRF failed, rc = 0x%x\n", rc);
} else {
    fprintf(stderr, "CKM_TLS_GOST_PRF test passed.\n");
}
}

if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
            "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr, "C_CloseSession success\n");
}

out:
    return rc;
}

CK_RV test_tls_prf(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    static CK_BYTE value[12];
    CK_MECHANISM mechanism_desc = {CKM_TLS_GOST_PRF, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;
    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_KEY_TYPE key_type = CKK_GENERIC_SECRET;
    static CK_BBOOL ltrue = CK_TRUE;
    // Using master secret (48 bytes) from LCJSSE test utility.
    static CK_BYTE keyval[] = {
        0x76, 0x22, 0x41, 0x3d, 0x5a, 0xb6, 0x7f, 0x5b,
        0x86, 0x2e, 0x75, 0x97, 0xe8, 0xf9, 0x31, 0xe4,
        0x2f, 0x13, 0x13, 0xd5, 0x61, 0xed, 0xab, 0xed,
        0xa5, 0x4b, 0xd8, 0x5d, 0x60, 0x0a, 0xec, 0x99,
        0x45, 0xfe, 0x39, 0xa1, 0xb7, 0x62, 0x0d, 0x66,
        0x3c, 0xdd, 0xb3, 0x71, 0x90, 0x0f, 0xc3, 0xdd
    };
    // Using real TLS message label from LirJSSE test utility.
    static CK_BYTE label[] = "client finished";

```

```
// seed (64 bytes) = client_random (32 bytes) + server_random
// (32 bytes)
// Using concatenated client_random and server_random data
// (32+32 bytes)
// from LCJSSE test utility.
static CK_BYTE seed [] = {
    0x87, 0x5a, 0x38, 0x94, 0xa4, 0xf9, 0x3c, 0x30,
    0x65, 0xfc, 0x66, 0xbe, 0xf2, 0xc0, 0x09, 0xb9,
    0xc3, 0x26, 0x3e, 0x16, 0xc7, 0x28, 0x14, 0x27,
    0x98, 0x26, 0xe4, 0xd5, 0x30, 0x7f, 0x9a, 0x2d,
    0x87, 0x0e, 0x1d, 0x34, 0xaf, 0x28, 0x1c, 0x60,
    0xae, 0x8d, 0x26, 0xe4, 0xd5, 0x30, 0x7f, 0x9a,
    0x2d, 0x87, 0x0e, 0x1d, 0x34, 0xaf, 0x28, 0x1c,
    0x60, 0xae, 0x8d, 0x21, 0xee, 0x7d, 0x52, 0x44
};
// Using resulting TLS PRF value (12 bytes) from LCJSSE test
// utility.
static CK_BYTE et_94 [] = {
    0x2a, 0x6b, 0x4c, 0x14, 0x61, 0xb1, 0x39, 0x2d,
    0x18, 0xdf, 0x8a, 0x8d,
};
static CK_BYTE et_2012_256 [] = {
    0x83, 0x75, 0x68, 0x86, 0x50, 0xb8, 0xe6, 0x5d,
    0x2c, 0x12, 0x45, 0xea,
};
static CK_BYTE et_2012_512 [] = {
    0x85, 0x4c, 0x6b, 0xb5, 0x8b, 0xdd, 0x9f, 0x8b,
    0xf2, 0xe3, 0xf4, 0x8e,
};
static CK_ATTRIBUTE key_template [] = {
    { CKA_VALUE, keyval, sizeof(keyval) },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_DERIVE, &ltrue, sizeof(ltrue) }
};
static CK_ULONG len = sizeof(value);

// В зависимости от значения OID для ГОСТ Р 34.11
// будут использоваться алгоритмы ГОСТ Р 34.11-94,
// ГОСТ Р 34.11-2012 (256 бит) или ГОСТ Р 34.11-2012 (512 бит)
.

static CK_BYTE oid_3411_94 [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
}
```

```
};
static CK_BYTE oid_3411_2012_256 [] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02
};
static CK_BYTE oid_3411_2012_512 [] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03
};
static CK_TLS_GOST_PRF_PARAMS params_gost = {
    {
        seed, sizeof(seed),
        label, sizeof(label) - 1,
        value, &len
    },
    oid_3411_94,
    sizeof(oid_3411_94)
};

rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

mechanism->pParameter = &params_gost;
mechanism->ulParameterLen = sizeof(params_gost);
rc = funcs->C_DeriveKey(sess, mechanism,
    keyh, NULL_PTR, 0, NULL_PTR);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed: 0x%x\n", rc);
    return rc;
}

printf("TLS PRF 3411-94:\n");
print_hex(value, len);
CHECK(value, len, et_94);

params_gost.pHashParamsOid = oid_3411_2012_256;
params_gost.ulHashParamsOidLen = sizeof(oid_3411_2012_256);
rc = funcs->C_DeriveKey(sess, mechanism,
    keyh, NULL_PTR, 0, NULL_PTR);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed: 0x%x\n", rc);
}
```

```

    return rc;
}

printf("TLS PRF 3411-2012 (256):\n");
print_hex(value, len);
CHECK(value, len, et_2012_256);

params_gost.pHashParamsOid = oid_3411_2012_512;
params_gost.ulHashParamsOidLen = sizeof(oid_3411_2012_512);
rc = funcs->C_DeriveKey(sess, mechanism,
    keyh, NULL_PTR, 0, NULL_PTR);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed: 0x%x\n", rc);
    return rc;
}

printf("TLS PRF 3411-2012 (512):\n");
print_hex(value, len);
CHECK(value, len, et_2012_512);
rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

return rc;
}

```

CKM_TLS_GOST_PRE_MASTER_KEY_GEN

Листинг 3.28: ckm_tls_gost_pre_master_key_gen.c

```

#include "test_common.h"

CK_RV test_crypto();
CK_RV test_tls_pre_master_key_gen(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
}

```

```

    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
    /*
    // log in as normal user
    rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
        user_pin));
    if (rc != CKR_OK) {
        fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
            rc);
        goto out_close;
    }
    fprintf(stderr, "C_Login success\n");
    */
    rc = funcs->C_GetMechanismInfo(SlotId,
        CKM_TLS_GOST_PRE_MASTER_KEY_GEN, &minfo);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "\n==== Mechanism CKM_TLS_GOST_PRE_MASTER_KEY_GEN "
            "not supported =====\n");
    } else {
        fprintf(stderr,
            "\n==== CKM_TLS_GOST_PRE_MASTER_KEY_GEN "

```



```

"test =====\n");
rc = test_tls_pre_master_key_gen(hSession);
if (rc != CKR_OK) {
    fprintf(stderr,
        "ERROR CKM_TLS_GOST_PRE_MASTER_KEY_GEN failed, "
        "rc = 0x%x\n", rc);
} else {
    fprintf(stderr,
        "CKM_TLS_GOST_PRE_MASTER_KEY_GEN test passed.\n");
}
}

if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr, "C_CloseSession success\n");
}
}

out:
return rc;
}

CK_RV test_tls_pre_master_key_gen(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[1024];
    CK_MECHANISM mechanism_desc =
        {CKM_TLS_GOST_PRE_MASTER_KEY_GEN, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;
    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_KEY_TYPE key_type = CKK_GENERIC_SECRET;
    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    CK_ATTRIBUTE attr;
    static CK_ATTRIBUTE ltemplate[] = {
        { CKA_CLASS, &oclass, sizeof(oclass) },
        { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
        { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
        { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) },
        { CKA_DERIVE, &ltrue, sizeof(ltrue) },
    }
}

```

```
{ CKA_SIGN, &ltrue, sizeof(ltrue) },
{ CKA_VERIFY, &ltrue, sizeof(ltrue) }
};
CK_VERSION ver; // TLS version

// В текущей версии всегда генерируется ключ типа
CKK_GENERIC_SECRET
// со случайным значением длиной 32 байта.
ver.major = 1;
ver.minor = 0;
mechanism->pParameter = &ver;
mechanism->ulParameterLen = sizeof(ver);
rc = funcs->C_GenerateKey(sess, mechanism,
    ltemplate, sizeof(ltemplate)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GenerateKey failed: 0x%x\n", rc);
    return rc;
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = 32;
rc = funcs->C_GetAttributeValue(sess, keyh, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    return rc;
}

printf("TLS PRE-MASTER KEY:\n");
print_hex(value, attr.ulValueLen);

rc = funcs->C_DestroyObject(sess, keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}

return rc;
}
```

CKM_TLS_GOST_MASTER_KEY_DERIVE

Листинг 3.29: ckm_tls_gost_master_key_derive.c

```
#include "test_common.h"

CK_RV test_crypto();
CK_RV test_tls_master_key_derive(CK_SESSION_HANDLE sess);

int main(int argc, char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc, argv);
    if (rc != CKR_OK) {
        fprintf(stderr, "test_main failed: 0x%x\n", rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr, "test_crypto failed: 0x%x\n", rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc, ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

    rc = funcs->C_OpenSession(SlotId,
        CKF_RW_SESSION | CKF_SERIAL_SESSION,
        NULL_PTR, NULL_PTR, &hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
        goto out;
    }
    fprintf(stderr, "C_OpenSession success\n");
    /*
    // log in as normal user
    rc = funcs->C_Login(hSession, CKU_USER, user_pin, strlen(
        user_pin));
    if (rc != CKR_OK) {
```

```

    fprintf(stderr, "ERROR call to C_Login failed, rc = 0x%x\n",
rc);
    goto out_close;
}
fprintf(stderr, "C_Login success\n");
*/
rc = funcs->C_GetMechanismInfo(SlotId,
    CKM_TLS_GOST_MASTER_KEY_DERIVE, &minfo);
if (rc != CKR_OK) {
    fprintf(stderr,
"\n===== Mechanism CKM_TLS_GOST_MASTER_KEY_DERIVE "
"not supported =====\n");
} else {
    fprintf(stderr,
"\n===== CKM_TLS_GOST_MASTER_KEY_DERIVE "
"test =====\n");
    rc = test_tls_master_key_derive(hSession);
    if (rc != CKR_OK) {
        fprintf(stderr,
            "ERROR CKM_TLS_GOST_MASTER_KEY_DERIVE failed, "
            "rc = 0x%x\n", rc);
    } else {
        fprintf(stderr,
            "CKM_TLS_GOST_MASTER_KEY_DERIVE test passed.\n");
    }
}

if ( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr, "C_CloseSession success\n");
}

out:
return rc;
}

CK_RV test_tls_master_key_derive(CK_SESSION_HANDLE sess)
{
    int rc = 0;
    CK_BYTE value[1024];

```

```

CK_ULONG len ;
CK_MECHANISM mechanism_desc =
    {CKM_TLS_GOST_MASTER_KEY_DERIVE, NULL, 0};
CK_MECHANISM_PTR mechanism = &mechanism_desc;
CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
CK_OBJECT_HANDLE mkeyh = CK_INVALID_HANDLE;
static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
static CK_KEY_TYPE key_type = CKK_GENERIC_SECRET;
static CK_BBOOL ltrue = CK_TRUE;
static CK_BBOOL lfalse = CK_FALSE;
CK_ATTRIBUTE attr;
FILE *f = NULL;
CK_OBJECT_HANDLE paramh = CK_INVALID_HANDLE;
static CK_BYTE premaster [] = {
    0xdc, 0x2f, 0x75, 0x53, 0x8a, 0x02, 0xc7, 0x4d,
    0x67, 0x32, 0x58, 0xf5, 0xf5, 0x27, 0xc3, 0xc9,
    0x27, 0xd6, 0x65, 0x8f, 0x72, 0x38, 0x7d, 0x4b,
    0x98, 0x06, 0xe0, 0x91, 0x1b, 0xa5, 0x8a, 0x30
};
static CK_BYTE client_random [] = {
    0x87, 0x5a, 0x38, 0x94, 0xa4, 0xf9, 0x3c, 0x30,
    0x65, 0xfc, 0x66, 0xbe, 0xf2, 0xc0, 0x09, 0xb9,
    0xc3, 0x26, 0x3e, 0x16, 0xc7, 0x28, 0x14, 0x27,
    0x98, 0x26, 0xe4, 0xd5, 0x30, 0x7f, 0x9a, 0x2d
};
static CK_BYTE server_random [] = {
    0x87, 0x0e, 0x1d, 0x34, 0xaf, 0x28, 0x1c, 0x60,
    0xae, 0x8d, 0x26, 0xe4, 0xd5, 0x30, 0x7f, 0x9a,
    0x2d, 0x87, 0x0e, 0x1d, 0x34, 0xaf, 0x28, 0x1c,
    0x60, 0xae, 0x8d, 0x21, 0xee, 0x7d, 0x52, 0x44
};
static CK_BYTE master_94 [] = {
    0x92, 0xf3, 0x91, 0xbc, 0xe7, 0x04, 0xe3, 0xbd,
    0x1c, 0x57, 0x6d, 0x8c, 0x55, 0x7b, 0x54, 0x2a,
    0x82, 0x52, 0x75, 0xfc, 0x79, 0x64, 0xcc, 0xcb,
    0xb3, 0x21, 0x41, 0xce, 0x50, 0x17, 0x23, 0x74,
    0xdd, 0xb9, 0xa1, 0xb3, 0x5b, 0xc2, 0x80, 0xab,
    0xd5, 0xec, 0x62, 0x70, 0x63, 0x6b, 0xcb, 0xb1,
};
static CK_BYTE master_2012_256 [] = {
    0x54, 0x27, 0x36, 0x05, 0x0b, 0x05, 0xa8, 0x47,
    0xc8, 0x98, 0xb2, 0xfc, 0x1e, 0xcc, 0xca, 0xbb,
    0x33, 0x58, 0x39, 0x78, 0xae, 0x82, 0x5c, 0xf2,
    0x71, 0xe2, 0x61, 0xce, 0xd7, 0x9a, 0x36, 0xc3,

```

```

    0x60, 0x9b, 0xea, 0x07, 0x01, 0x52, 0x1d, 0x87,
    0x5a, 0x93, 0x16, 0x28, 0xb0, 0x30, 0xd5, 0x32,
};
static CK_BYTE master_2012_512[] = {
    0x7c, 0xcf, 0xa7, 0x35, 0x3d, 0xb2, 0x14, 0x75,
    0x43, 0x0c, 0xe8, 0x7c, 0x0d, 0x25, 0x84, 0x66,
    0x07, 0x4d, 0x06, 0x94, 0xa3, 0x66, 0x01, 0xf5,
    0x74, 0xe0, 0xeb, 0x03, 0x1b, 0x98, 0x84, 0xd1,
    0xc5, 0x19, 0xcc, 0xa6, 0x3d, 0x33, 0xb7, 0x05,
    0x1e, 0xd4, 0x9c, 0x9e, 0x77, 0x0f, 0xe6, 0x2b,
};
// В зависимости от значения OID для ГОСТ Р 34.11
// в функции TLS PRF будут использоваться алгоритмы ГОСТ Р
// 34.11-94,
// ГОСТ Р 34.11-2012 (256 бит) или ГОСТ Р 34.11-2012 (512 бит)
.
static CK_BYTE oid_3411_94[] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
};
static CK_BYTE oid_3411_2012_256[] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02
};
static CK_BYTE oid_3411_2012_512[] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03
};
static CK_ATTRIBUTE key_template[] = {
    { CKA_VALUE, premaster, sizeof(premaster) },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &key_type, sizeof(key_type) },
    { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) },
    { CKA_DERIVE, &ltrue, sizeof(ltrue) },
    { CKA_SIGN, &ltrue, sizeof(ltrue) },
    { CKA_VERIFY, &ltrue, sizeof(ltrue) }
};
static CK_TLS_GOST_MASTER_KEY_DERIVE_PARAMS params_gost = {
    { client_random, sizeof(client_random),
    server_random, sizeof(server_random) }, // RandomInfo
    oid_3411_94, // pHashParamsOid
    sizeof(oid_3411_94) // ulHashParamsOidLen
};

rc = funcs->C_CreateObject(sess,
    key_template, sizeof(key_template)/sizeof(CK_ATTRIBUTE),

```

```
&keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    goto end;
}

mechanism->pParameter = &params_gost;
mechanism->ulParameterLen = sizeof(params_gost);

rc = funcs->C_DeriveKey(sess, mechanism, keyh,
    key_template + 1,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE) - 1,
    &mkeyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed: 0x%x\n", rc);
    goto end;
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, mkeyh, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    goto end;
}
len = attr.ulValueLen;

printf("TLS master key 3411-94:\n");
print_hex(value, len);
CHECK(value, len, master_94);

funcs->C_DestroyObject(sess, mkeyh);

params_gost.pHashParamsOid = oid_3411_2012_256;
params_gost.ulHashParamsOidLen = sizeof(oid_3411_2012_256);

rc = funcs->C_DeriveKey(sess, mechanism, keyh,
    key_template + 1,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE) - 1,
    &mkeyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed: 0x%x\n", rc);
    goto end;
}
```

```
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, mkeyh, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "C_GetAttributeValue failed: 0x%x\n", rc);
    goto end;
}
len = attr.ulValueLen;

printf("TLS master key 3411-2012 (256):\n");
print_hex(value, len);
CHECK(value, len, master_2012_256);

funcs->C_DestroyObject(sess, mkeyh);

params_gost.pHashParamsOid = oid_3411_2012_512;
params_gost.ulHashParamsOidLen = sizeof(oid_3411_2012_512);

rc = funcs->C_DeriveKey(sess, mechanism, keyh,
    key_template + 1,
    sizeof(key_template)/sizeof(CK_ATTRIBUTE) - 1,
    &mkeyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed: 0x%x\n", rc);
    goto end;
}

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess, mkeyh, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr,
        "C_GetAttributeValue failed: 0x%x\n", rc);
    goto end;
}
len = attr.ulValueLen;

printf("TLS master key 3411-2012 (512):\n");
print_hex(value, len);
```



```

CHECK(value , len , master_2012_512);

funcs->C_DestroyObject(sess , mkeyh);

printf("SUCCESS\n");
rc = CKR_OK;
end:
if (keyh != CK_INVALID_HANDLE) {
    funcs->C_DestroyObject(sess , keyh);
}

return rc;
}

```

CKM_TLS_GOST_KEY_AND_MAC_DERIVE

Листинг 3.30: ckm_tls_gost_key_and_mac_derive.c

```

#include "test_common.h"
CK_RV test_crypto();
CK_RV test_tls_key_and_mac_derive(CK_SESSION_HANDLE sess);

int main(int argc , char *argv[]) {
    CK_RV rc = CKR_OK;

    rc = test_main(argc , argv);
    if (rc != CKR_OK) {
        fprintf(stderr , "test_main failed: 0x%x\n" , rc);
        return rc;
    }
    rc = test_crypto();
    if (rc != CKR_OK) {
        fprintf(stderr , "test_crypto failed: 0x%x\n" , rc);
        return rc;
    }
    return CKR_OK;
}

CK_RV test_crypto()
{
    CK_RV rc , ret;
    CK_SESSION_HANDLE hSession;
    CK_MECHANISM_INFO minfo;

```

```

rc = funcs->C_OpenSession( SlotId ,
    CKF_RW_SESSION | CKF_SERIAL_SESSION,
    NULL_PTR, NULL_PTR, &hSession);
if (rc != CKR_OK) {
    fprintf(stderr ,
        "ERROR call to C_OpenSession failed, rc = 0x%x\n", rc);
    goto out;
}
fprintf(stderr , "C_OpenSession success\n");
rc = funcs->C_GetMechanismInfo( SlotId ,
    CKM_TLS_GOST_KEY_AND_MAC_DERIVE, &minfo);
if (rc != CKR_OK) {
    fprintf(stderr ,
"\n===== Mechanism CKM_TLS_GOST_KEY_AND_MAC_DERIVE "
"not supported =====\n");
} else {
    fprintf(stderr ,
"\n===== CKM_TLS_GOST_KEY_AND_MAC_DERIVE "
"test =====\n");
    rc = test_tls_key_and_mac_derive(hSession);
    if (rc != CKR_OK) {
        fprintf(stderr ,
            "ERROR CKM_TLS_GOST_KEY_AND_MAC_DERIVE failed, "
            "rc = 0x%x\n", rc);
    } else {
        fprintf(stderr ,
            "CKM_TLS_GOST_KEY_AND_MAC_DERIVE test passed.\n");
    }
}

if( (ret = funcs->C_CloseSession(hSession)) != CKR_OK ) {
    fprintf(stderr ,
        "Error: C_CloseSession failed with 0x%x\n", ret);
    rc = ret;
}
else {
    fprintf(stderr , "C_CloseSession success\n");
}

out:
return rc;
}

```

```
CK_RV test_tls_key_and_mac_derive(CK_SESSION_HANDLE sess)
```

```

{
    int rc = 0;
    CK_BYTE value[1024];
    CK_ULONG len;
    CK_MECHANISM mechanism_desc =
    {CKM_TLS_GOST_KEY_AND_MAC_DERIVE, NULL, 0};
    CK_MECHANISM_PTR mechanism = &mechanism_desc;

    CK_OBJECT_HANDLE keyh = CK_INVALID_HANDLE;
    static CK_OBJECT_CLASS oclass = CKO_SECRET_KEY;
    static CK_KEY_TYPE generic_key_type = CKK_GENERIC_SECRET;
    static CK_KEY_TYPE secret_key_type = CKK_GOST28147;
    static CK_BBOOL ltrue = CK_TRUE;
    static CK_BBOOL lfalse = CK_FALSE;
    CK_ATTRIBUTE attr;

    CK_OBJECT_HANDLE paramh = CK_INVALID_HANDLE;

    static CK_BYTE client_random[] = {
        0x48, 0xdc, 0xdb, 0x79, 0xfb, 0x11, 0x16, 0xaf,
        0xed, 0x6a, 0x72, 0xda, 0xbf, 0x7a, 0xb2, 0x76,
        0x8c, 0x35, 0xa7, 0x54, 0x52, 0xda, 0xa1, 0x00,
        0x47, 0x83, 0x29, 0x72, 0xa9, 0x8d, 0xd9, 0x78
    };
    static CK_BYTE server_random[] = {
        0x48, 0xdc, 0xda, 0xaa, 0x83, 0xeb, 0x5c, 0x08,
        0x51, 0x20, 0xc6, 0x8e, 0x29, 0xb4, 0x30, 0xff,
        0xc5, 0x9e, 0x4f, 0x1c, 0xa9, 0x75, 0xa7, 0x35,
        0x62, 0x89, 0x29, 0x17, 0x28, 0x70, 0x56, 0x06
    };
    static CK_BYTE master[] = {
        0x68, 0x0d, 0x31, 0x53, 0x6a, 0x20, 0x68, 0x50,
        0xca, 0x66, 0x01, 0x3f, 0xb6, 0xfa, 0xe3, 0x26,
        0x51, 0xb5, 0xba, 0x03, 0x16, 0xa3, 0xdf, 0xc1,
        0x33, 0x99, 0xd0, 0x61, 0xda, 0x74, 0x12, 0x4e,
        0x96, 0x6f, 0x9a, 0x1c, 0x25, 0x24, 0x89, 0x42,
        0xa9, 0xeb, 0x0c, 0x56, 0xe1, 0x04, 0xbb, 0x6a
    };
    static CK_BYTE ClientMacSecret_94[] = {
        0x12, 0x9d, 0xf5, 0xe1, 0x00, 0xfe, 0x51, 0xf5,
        0xe9, 0xf3, 0xcf, 0xff, 0x4f, 0x02, 0xcc, 0xc9,
        0x4a, 0xe8, 0x32, 0x11, 0xbd, 0x35, 0x63, 0xa8,
        0xe2, 0xa9, 0x71, 0xd2, 0x61, 0x24, 0x7b, 0x23
    };
};

```

```
static CK_BYTE ServerMacSecret_94 [] = {
    0xc0, 0xfc, 0x04, 0x6f, 0xa2, 0xc6, 0x30, 0x97,
    0xa2, 0x90, 0x8a, 0x47, 0x56, 0x26, 0x9d, 0xc9,
    0xe9, 0x87, 0x71, 0x41, 0xc4, 0x3d, 0x81, 0x81,
    0xb0, 0x30, 0x8f, 0xea, 0xa3, 0x86, 0x4f, 0xae
};
static CK_BYTE ClientKey_94 [] = {
    0x0e, 0xfc, 0x85, 0x03, 0x2b, 0x40, 0xbd, 0x65,
    0xce, 0x39, 0x70, 0xb0, 0xed, 0xd2, 0x48, 0xdd,
    0x6e, 0xfc, 0x97, 0x29, 0xb4, 0xd5, 0xd8, 0x91,
    0x42, 0x7a, 0xd0, 0x55, 0x9c, 0x7c, 0x6e, 0x61
};
static CK_BYTE ServerKey_94 [] = {
    0x17, 0x34, 0xe4, 0x2c, 0x8e, 0x9c, 0x6f, 0xda,
    0x3c, 0x30, 0xbb, 0xc8, 0xfa, 0xe7, 0x8d, 0x76,
    0x11, 0x0c, 0xfc, 0x5b, 0xeb, 0x00, 0x92, 0xb2,
    0xed, 0x36, 0x3e, 0x27, 0x60, 0x12, 0x50, 0xa5
};
static CK_BYTE IVs_94 [] = {
    0x99, 0x38, 0xb5, 0x71, 0x78, 0x47, 0x92, 0xba,
    0xc4, 0x43, 0xb9, 0xd3, 0x24, 0x75, 0x09, 0x79
};
static CK_BYTE ClientMacSecret_2012_256 [] = {
    0x14, 0xe7, 0xfc, 0x31, 0x57, 0x99, 0x80, 0x9d,
    0x48, 0x98, 0xd6, 0x43, 0x88, 0xfc, 0x1a, 0xa4,
    0x8c, 0x21, 0x1d, 0x38, 0x72, 0x0d, 0x71, 0xcf,
    0x31, 0x41, 0x95, 0xff, 0x4c, 0xda, 0x74, 0xa9,
};
static CK_BYTE ServerMacSecret_2012_256 [] = {
    0xa0, 0x6e, 0xc5, 0x6f, 0xa9, 0x66, 0x06, 0xfa,
    0x7b, 0x20, 0x83, 0xb9, 0xaa, 0xc3, 0xc2, 0xb2,
    0xd1, 0x73, 0x2a, 0x89, 0x99, 0xeb, 0x29, 0xe3,
    0xf7, 0xa4, 0x1e, 0x5c, 0x22, 0x21, 0xf8, 0xc0,
};
static CK_BYTE ClientKey_2012_256 [] = {
    0x6b, 0x42, 0x50, 0x6a, 0x7d, 0xc7, 0x0b, 0x82,
    0x5d, 0xe5, 0x63, 0x60, 0x16, 0x8c, 0xbe, 0x8f,
    0x70, 0x1d, 0x74, 0x1a, 0x99, 0x3b, 0x17, 0x66,
    0x67, 0xbc, 0x81, 0x9e, 0x44, 0x2f, 0xe0, 0xac,
};
static CK_BYTE ServerKey_2012_256 [] = {
    0x95, 0x45, 0xbd, 0xff, 0x4b, 0x12, 0x61, 0x3c,
    0xf5, 0x81, 0xcd, 0x3a, 0xd9, 0xcf, 0xef, 0x4e,
    0x38, 0x00, 0xd3, 0xbd, 0xc8, 0x03, 0x91, 0xcf,
```

```
    0x4b, 0x06, 0xe5, 0xa1, 0x4f, 0x8d, 0x61, 0x9e,
};
static CK_BYTE IVs_2012_256[] = {
    0x05, 0x49, 0xbf, 0xbc, 0x7f, 0x79, 0xc3, 0x38,
    0x41, 0xb1, 0x9e, 0x02, 0x6c, 0x61, 0x77, 0xa8,
};
static CK_BYTE ClientMacSecret_2012_512[] = {
    0x6b, 0x19, 0x9f, 0xf5, 0x37, 0x80, 0x16, 0xc1,
    0xc1, 0x67, 0x92, 0x77, 0xad, 0x22, 0xb9, 0xd2,
    0x95, 0x1c, 0xac, 0x40, 0xeb, 0x44, 0x47, 0x9f,
    0xe5, 0xe0, 0xa0, 0x09, 0x0a, 0xe1, 0xad, 0x65,
};
static CK_BYTE ServerMacSecret_2012_512[] = {
    0x93, 0x25, 0x4c, 0x12, 0xa6, 0x39, 0x6e, 0xf7,
    0x34, 0xeb, 0x6c, 0x47, 0x53, 0x41, 0x36, 0x0e,
    0x06, 0xd7, 0x19, 0x46, 0x69, 0x36, 0x5d, 0x6a,
    0xf5, 0xae, 0xe6, 0x89, 0xf4, 0x45, 0xc2, 0x73,
};
static CK_BYTE ClientKey_2012_512[] = {
    0xc4, 0x3b, 0x32, 0x78, 0xb0, 0x75, 0xcd, 0x7f,
    0x34, 0xa4, 0x98, 0xe3, 0xad, 0x1d, 0x9c, 0x0a,
    0x69, 0x7d, 0xc0, 0x40, 0x79, 0x42, 0xfd, 0x69,
    0x31, 0xec, 0x87, 0x6f, 0x05, 0xae, 0x88, 0x8e,
};
static CK_BYTE ServerKey_2012_512[] = {
    0x4c, 0x7f, 0xad, 0xa0, 0xb6, 0xb9, 0x8f, 0xda,
    0xff, 0x6d, 0xb8, 0xec, 0x13, 0x31, 0x54, 0xd7,
    0x6c, 0xfb, 0xfc, 0x0e, 0xd3, 0xdd, 0xe0, 0xf0,
    0x89, 0x48, 0x8e, 0x8a, 0x5d, 0x96, 0x65, 0x71,
};
static CK_BYTE IVs_2012_512[] = {
    0x6d, 0x21, 0xda, 0x52, 0xaa, 0xcf, 0x84, 0x01,
    0x4d, 0x94, 0xec, 0x24, 0x6a, 0x17, 0xeb, 0x05,
};

// В зависимости от значения OID для ГОСТ Р 34.11
// в функции TLS PRF будут использоваться алгоритмы ГОСТ Р
// 34.11-94,
// ГОСТ Р 34.11-2012 (256 бит) или ГОСТ Р 34.11-2012 (512 бит)
.
static CK_BYTE oid_3411_94[] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1e, 0x01
};
static CK_BYTE oid_3411_2012_256[] = {
```

```
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02
};
static CK_BYTE oid_3411_2012_512 [] = {
    0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03
};

static CK_BYTE gost28147_params_A [] = {
    0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01
};

static CK_ATTRIBUTE master_key_template [] = {
    { CKA_VALUE, master, sizeof(master) },
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE,
    &generic_key_type, sizeof(generic_key_type) },
    { CKA_DERIVE, &ltrue, sizeof(ltrue) },
    { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) }
};

static CK_ATTRIBUTE secret_key_template [] = {
    { CKA_CLASS, &oclass, sizeof(oclass) },
    { CKA_KEY_TYPE, &secret_key_type, sizeof(secret_key_type) },
    { CKA_GOST28147_PARAMS,
    gost28147_params_A, sizeof(gost28147_params_A) },
    { CKA_ENCRYPT, &ltrue, sizeof(ltrue) },
    { CKA_DECRYPT, &ltrue, sizeof(ltrue) },
    { CKA_SIGN, &ltrue, sizeof(ltrue) },
    { CKA_VERIFY, &ltrue, sizeof(ltrue) },
    { CKA_SENSITIVE, &lfalse, sizeof(lfalse) },
    { CKA_EXTRACTABLE, &ltrue, sizeof(ltrue) }
};

static CK_BYTE IVClient [8];
static CK_BYTE IVServer [8];
static CK_SSL3_KEY_MAT_OUT km = {
    CK_INVALID_HANDLE, CK_INVALID_HANDLE,
    CK_INVALID_HANDLE, CK_INVALID_HANDLE,
    IVClient, IVServer
};

static CK_TLS_GOST_KEY_MAT_PARAMS params_gost =
{
    {
        256, 256, 64, CK_FALSE,
```

```
        {
            client_random, sizeof(client_random),
            server_random, sizeof(server_random)
        },
        &km
    },
    oid_3411_94,
    sizeof(oid_3411_94)
};

rc = funcs->C_CreateObject(sess,
    master_key_template,
    sizeof(master_key_template)/sizeof(CK_ATTRIBUTE),
    &keyh);
if (rc != CKR_OK) {
    fprintf(stderr, "C_CreateObject failed: 0x%x\n", rc);
    return rc;
}

mechanism->pParameter = &params_gost;
mechanism->ulParameterLen = sizeof(params_gost);

rc = funcs->C_DeriveKey(sess, mechanism, keyh,
    secret_key_template,
    sizeof(secret_key_template)/sizeof(CK_ATTRIBUTE),
    NULL_PTR);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed: 0x%x\n", rc);
    return rc;
}
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess,
    km.hClientMacSecret, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    return rc;
}
len = attr.ulValueLen;

printf("ClientMacSecret 3411-94:\n");
print_hex(value, len);
```

```
CHECK(value , len , ClientMacSecret_94);

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess ,
    km.hServerMacSecret , &attr , 1);
if (rc != CKR_OK) {
    fprintf(stderr , "C_GetAttributeValue failed: 0x%x\n" , rc);
    return rc;
}
len = attr.ulValueLen;

printf("ServerMacSecret 3411-94:\n");
print_hex(value , len);
CHECK(value , len , ServerMacSecret_94);

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess ,
    km.hClientKey , &attr , 1);
if (rc != CKR_OK) {
    fprintf(stderr , "C_GetAttributeValue failed: 0x%x\n" , rc);
    return rc;
}
len = attr.ulValueLen;

printf("ClientKey 3411-94:\n");
print_hex(value , len);
CHECK(value , len , ClientKey_94);

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess ,
    km.hServerKey , &attr , 1);
if (rc != CKR_OK) {
    fprintf(stderr , "C_GetAttributeValue failed: 0x%x\n" , rc);
    return rc;
}
len = attr.ulValueLen;

printf("ServerKey 3411-94:\n");
```



```
print_hex(value , len);
CHECK(value , len , ServerKey_94);

memcpy(value , IVClient , 8);
memcpy(value + 8 , IVServer , 8);
len = 16;

printf("IVClient & IVServer 3411-94:\n");
print_hex(value , len);
CHECK(value , len , IVs_94);

rc = funcs->C_DestroyObject(sess , km.hClientMacSecret);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , km.hServerMacSecret);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , km.hClientKey);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , km.hServerKey);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}

params_gost.pHashParamsOid = oid_3411_2012_256;
params_gost.ulHashParamsOidLen = sizeof(oid_3411_2012_256);

rc = funcs->C_DeriveKey(sess , mechanism , keyh ,
    secret_key_template ,
    sizeof(secret_key_template)/sizeof(CK_ATTRIBUTE) ,
    NULL_PTR);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DeriveKey failed: 0x%x\n" , rc);
    return rc;
}
attr.type = CKA_VALUE;
```

```
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess,
    km.hClientMacSecret, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    return rc;
}
len = attr.ulValueLen;

printf("ClientMacSecret 3411-2012 (256):\n");
print_hex(value, len);
CHECK(value, len, ClientMacSecret_2012_256);

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess,
    km.hServerMacSecret, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    return rc;
}
len = attr.ulValueLen;

printf("ServerMacSecret 3411-2012 (256):\n");
print_hex(value, len);
CHECK(value, len, ServerMacSecret_2012_256);

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess,
    km.hClientKey, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    return rc;
}
len = attr.ulValueLen;

printf("ClientKey 3411-2012 (256):\n");
print_hex(value, len);
CHECK(value, len, ClientKey_2012_256);
```

```
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess,
    km.hServerKey, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    return rc;
}
len = attr.ulValueLen;

printf("ServerKey 3411-2012 (256):\n");
print_hex(value, len);
CHECK(value, len, ServerKey_2012_256);

memcpy(value, IVClient, 8);
memcpy(value + 8, IVServer, 8);
len = 16;

printf("IVClient & IVServer 3411-2012 (256):\n");
print_hex(value, len);
CHECK(value, len, IVs_2012_256);

rc = funcs->C_DestroyObject(sess, km.hClientMacSecret);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess, km.hServerMacSecret);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess, km.hClientKey);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess, km.hServerKey);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DestroyObject failed: 0x%x\n", rc);
    return rc;
}
}
```

```
params_gost.pHashParamsOid = oid_3411_2012_512;
params_gost.ulHashParamsOidLen = sizeof(oid_3411_2012_512);

rc = funcs->C_DeriveKey(sess, mechanism, keyh,
    secret_key_template,
    sizeof(secret_key_template)/sizeof(CK_ATTRIBUTE),
    NULL_PTR);
if (rc != CKR_OK) {
    fprintf(stderr, "C_DeriveKey failed: 0x%x\n", rc);
    return rc;
}
attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess,
    km.hClientMacSecret, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    return rc;
}
len = attr.ulValueLen;

printf("ClientMacSecret 3411-2012 (512):\n");
print_hex(value, len);
CHECK(value, len, ClientMacSecret_2012_512);

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess,
    km.hServerMacSecret, &attr, 1);
if (rc != CKR_OK) {
    fprintf(stderr, "C_GetAttributeValue failed: 0x%x\n", rc);
    return rc;
}
len = attr.ulValueLen;

printf("ServerMacSecret 3411-2012 (512):\n");
print_hex(value, len);
CHECK(value, len, ServerMacSecret_2012_512);

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
```

```
rc = funcs->C_GetAttributeValue(sess ,
    km.hClientKey , &attr , 1);
if (rc != CKR_OK) {
    fprintf(stderr , "C_GetAttributeValue failed: 0x%x\n" , rc);
    return rc;
}
len = attr.ulValueLen;

printf("ClientKey 3411-2012 (512):\n");
print_hex(value , len);
CHECK(value , len , ClientKey_2012_512);

attr.type = CKA_VALUE;
attr.pValue = value;
attr.ulValueLen = sizeof(value);
rc = funcs->C_GetAttributeValue(sess ,
    km.hServerKey , &attr , 1);
if (rc != CKR_OK) {
    fprintf(stderr , "C_GetAttributeValue failed: 0x%x\n" , rc);
    return rc;
}
len = attr.ulValueLen;

printf("ServerKey 3411-2012 (512):\n");
print_hex(value , len);
CHECK(value , len , ServerKey_2012_512);

memcpy(value , IVClient , 8);
memcpy(value + 8 , IVServer , 8);
len = 16;

printf("IVClient & IVServer 3411-2012 (512):\n");
print_hex(value , len);
CHECK(value , len , IVs_2012_512);

rc = funcs->C_DestroyObject(sess , km.hClientMacSecret);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , km.hServerMacSecret);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}
```

```
}
rc = funcs->C_DestroyObject(sess , km.hClientKey);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}
rc = funcs->C_DestroyObject(sess , km.hServerKey);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}

rc = funcs->C_DestroyObject(sess , keyh);
if (rc != CKR_OK) {
    fprintf(stderr , "C_DestroyObject failed: 0x%x\n" , rc);
    return rc;
}
return rc;
}
```

3.3.19 Использование нескольких потоков

В программе `threadmkobj.c` производится вызов функций PKCS#11 из нескольких потоков.

Обратите внимание, что при инициализации библиотеки в аргументе функции `C_Initialize` (внутри функции `init`) приложением передаются адреса функций, используемых в библиотеке для блокировки потоков. Согласно стандарту, допускается использование блокировочных функций приложения или библиотеки. Если бы приложение работало с библиотекой в одном потоке, то функцию `C_Initialize` нужно было бы вызывать с аргументом `NULL`, как это делается в большинстве тестовых примеров.

Листинг 3.31: `treadmkobj.c`

```
#include "test_common.h"

// Количество приватных объектов на токене не должно превышать
// 256,
// количество публичных объектов на токене тоже не должно превышать
// 256,
// количество одновременно открытых сессий не должно превышать
// 256,
// а количество запускаемых потоков ограничено операционной сист
// емой,
```

```

// поэтому задавать здесь произвольно большие значения не рекомендуе
тся.
#define THREADCNT 4 // количество потоков (максимум зависит от с
истемы)
#define NUMSESS 3 // количество сессий в потоке
#define NUMOBJ 5 // максимальное количество объектов в сессии

CK_RV
open_session_and_login( CK_SLOT_ID SlotID )
{
    CK_FLAGS          flags;
    CK_SESSION_HANDLE h_session;
    CK_RV             rc;
    CK_BYTE           ltrue  = TRUE;

    flags = CKF_SERIAL_SESSION | CKF_RW_SESSION;
    rc = funcs->C_OpenSession(
        SlotID, flags, NULL, NULL, &h_session );

    rc = funcs->C_Login( h_session, CKU_USER,
        user_pin, strlen( user_pin ) );

    return rc;
}

int create_token_objects( int id, int is )
{
    CK_FLAGS          flags;
    CK_SESSION_HANDLE h_session;
    CK_RV             rc;
    CK_BYTE           ltrue  = TRUE;
    CK_BYTE           lfalse = FALSE;
    CK_OBJECT_CLASS   data_class      = CKO_DATA;
    CK_UTF8CHAR       label[32];
    CK_CHAR            app[] = "ls_hw11_library";
    CK_BYTE            data_value[]
        = "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz";
    CK_ATTRIBUTE       data_attribs[] =
    {
        {CKA_CLASS,      &data_class,  sizeof(data_class)},
        {CKA_APPLICATION, app,          strlen(app)+1},
        {CKA_TOKEN,      &ltrue,        sizeof(ltrue)},
        {CKA_VALUE,      &data_value,  sizeof(data_value)},
    }
}

```

```

    {CKA_PRIVATE, &lfalse, sizeof(lfalse)}
};
CK_ATTRIBUTE label_attribs[] =
{
    {CKA_LABEL, label, 0},
};
CK_OBJECT_HANDLE obj_list[NUMOBJ];
CK_ULONG i = 0;
CK_ULONG count = NUMOBJ;

flags = CKF_SERIAL_SESSION | CKF_RW_SESSION;
rc = funcs->C_OpenSession(
    SlotId, flags, NULL, NULL, &h_session );
if (rc != CKR_OK) {
    show_error(" C_OpenSession", rc );
    goto done;
}
fprintf(stderr, "Thread %d: Session %d opened\n", id, is);

// В каждой сессии мы выполняем операции создания,
// модификации и уничтожения случайного количества
// объектов токена, чтобы создать конкурентную обстановку
// между потоками.

rc = funcs->C_GenerateRandom(h_session,
    (CK_BYTE *)&count, sizeof(count));
if (rc != CKR_OK) {
    show_error(" C_GenerateRandom", rc );
    goto done;
}
count = count % NUMOBJ + 1;

for (i=0; i<count; i++) {
    rc = funcs->C_CreateObject( h_session,
        data_attribs, sizeof(data_attribs)/sizeof(CK_ATTRIBUTE),
        &obj_list[i] );
    if (rc != CKR_OK) {
        show_error(" C_CreateObject", rc );
        goto done;
    }

    sprintf(label, "data_%d_%d_%d", id, is, i+1);
    label_attribs[0].ulValueLen = strlen(label)+1;
}

```



```

rc = funcs->C_SetAttributeValue( h_session , obj_list[i] ,
    label_attr , sizeof(label_attr)/sizeof(CK_ATTRIBUTE) )
;
if (rc != CKR_OK) {
    show_error("    C_SetAttributeValue", rc );
    goto done;
}

fprintf(stderr , "Object %s created\n", label);
}
fprintf(stderr ,
    "Thread %d: %d objects of session %d created\n",
    id , count , is );

for (i=0; i<count; i++) {
    rc = funcs->C_DestroyObject( h_session , obj_list[i] );
    if (rc != CKR_OK) {
        show_error("    C_DestroyObject", rc );
        goto done;
    }
    sprintf(label , "data_%d_%d_%d", id , is , i+1);
    fprintf(stderr , "Object %s destroyed\n", label);
}
fprintf(stderr ,
    "Thread %d: %d objects of %d destroyed\n",
    id , count , is );

done:

funcs->C_CloseSession( h_session );

fprintf(stderr , "Session %d closed\n", id);

return rc;
}

#ifdef WIN32
DWORD WINAPI
#else
int
#endif
thread_func(void *thid)
{
    int    i=0;

```

```

CK_RV rv = CKR_OK;
int id = *(int *)thid;

fprintf(stderr, "thread_func %d started\n", id);
while(i++ < NUMSESS) {
    fprintf(stderr,
        "create_token_objects %d, session %d\n", id, i);
    rv = create_token_objects(id, i);
    if (rv != CKR_OK) {
        fprintf(stderr,
            "create_token_objects %d, session %d failed\n",
            id, i);
        break;
    }
    fprintf(stderr,
        "create_token_objects %d, session %d finished\n",
        id, i);
}

fprintf(stderr, "thread_func %d finished\n", id);
return rv;
}

int
main( int argc, char **argv )
{
    int i, rc;
#ifdef WIN32
    HANDLE id [THREADCNT];
    DWORD rv;
#else
    pthread_t id [THREADCNT];
#endif
    int thid [THREADCNT];

    for (i=1; i < argc; i++) {
        if (strcmp(argv[i], "-api") == 0) {
            i++;
            api_path = argv[i];
        } else if (strcmp(argv[i], "-user_pin") == 0) {
            i++;
            user_pin = argv[i];
        } else if (strcmp(argv[i], "-h") == 0) {
            printf(

```

```
        "usage:  %s [-api <path>][-user_pin <PIN>] [-h]\n\n",
        argv[0] );
    return 0;
}
}

fprintf(stderr, "==>> init...\n");
// Use PKCS#11 library native multithreading locks.
rc = init(api_path, &funcs, pinit_args_os_locking);
// Use application-supplied multithreading callbacks for
locking.
//rc = init(api_path, &funcs, pinit_args_app_locking);
if (rc != CKR_OK) {
    fprintf(stderr, "==>> init failed\n\n");
    return rc;
}
fprintf(stderr, "==>> init Ok\n\n");

fprintf(stderr, "==>> get_slot_list...\n");
rc = get_slot_list(funcs,
                  &SlotList,
                  &SlotCount);
if (rc != CKR_OK) {
    fprintf(stderr,
        "==>> get_slot_list failed, rc = 0x%x\n\n", rc );
    return rc;
}
fprintf(stderr, "==>> get_slot_list Ok\n\n");

fprintf(stderr, "==>> find_slot_with_token...\n");
rc = find_slot_with_token(funcs,
                          SlotList,
                          SlotCount,
                          &SlotId);
if (rc != CKR_OK) {
    fprintf(stderr,
        "==>> find_slot_with_token failed, rc = 0x%x\n\n",
        rc );
    return rc;
}
fprintf(stderr, "==>> find_slot_with_token Ok\n\n");

rc = open_session_and_login(SlotId);
if (rc != CKR_OK) {
```

```
fprintf(stderr ,
    "=>> open_session_and_login failed, rc = 0x%x\n\n",
    rc );
return rc;
}
fprintf(stderr , "=>> open_session_and_login Ok\n\n");

for (i=0;i<THREADCNT;i++){
    thid[i] = i+1;
    fprintf(stderr , "Creating thread %d \n", thid[i]);
#ifdef WIN32
    id[i] = CreateThread(NULL, 0,
        (LPTHREAD_START_ROUTINE)thread_func ,
        (void *)&(thid[i]), 0, NULL);
#else
    pthread_create(&id[i],NULL,
        (void*)(*)(void *)thread_func ,(void *)&(thid[i]));
#endif
}

fprintf(stderr , "Waiting for all application threads\n");
#ifdef WIN32
rv = WaitForMultipleObjects(THREADCNT, id , TRUE, INFINITE);
if (rv == WAIT_TIMEOUT) {
    fprintf(stderr , "Stop waiting by timeout\n");
} else {
    fprintf(stderr , "All threads completed\n");
}
#else
for (i=0; i<THREADCNT; i++){
    if (thid[i]) {
        printf("Joining thread %ld\n", thid[i]);
        pthread_join(id[i],NULL);
        thid[i] = 0;
    }
}
#endif

fprintf(stderr ,
    "Finalizing library...\n");
funcs->C_Finalize(NULL);
fprintf(stderr , "SUCCESS\n");
return 0;
}
```


4 Лицензии

4.1 Лицензия для BigDigits

В библиотеке ls11sw2012 используется библиотека BigDigits [??] 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 2.2 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.

5 Ссылки

1. Официальный сайт ООО "ЛИССИ-Софт". - <http://http://soft.lissi.ru/>.
2. Официальный сайт Технического комитета по стандартизации (ТК 26) "Криптографическая защита информации". - <https://www.tc26.ru>.
3. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. – <http://protect.gost.ru/document.aspx?control=7&id=139177>.
4. 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>.
5. ГОСТ Р 34.10-2012. Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи. – Москва, Стандартинформ, 2012.
6. ГОСТ Р 34.11-94. Информационная технология. Криптографическая защита информации. Функция хеширования. – <http://protect.gost.ru/document.aspx?control=7&id=134550>.
7. ГОСТ Р 34.11-2012. Информационная технология. Криптографическая защита информации. Функция хеширования. – Москва, Стандартинформ, 2012.
8. 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>.
9. 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>.
10. Методические рекомендации по заданию узлов замены блока подстановки алгоритма шифрования ГОСТ 28147-89 (готовится к публикации). – Москва, ТК 26, 2013.

11. Методические рекомендации по заданию параметров эллиптических кривых в соответствии с ГОСТ Р 34.10-2012 (готовится к публикации). – Москва, ТК 26, 2013.
12. 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.
13. Парольная защита с использованием алгоритмов ГОСТ. Дополнения к PKCS#5 (готовится к публикации). – Москва, ТК 26, 2013.
14. Расширение PKCS#11 для использования российских криптографических алгоритмов. – Технический комитет по стандартизации (ТК 26) "Криптографическая защита информации". – Москва, ТК 26, 2008.
15. 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.
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/>.