

# Встраивание устройств Рутокен через PKCS#11

- 1 Введение
  - 1.1 Рутокен S
  - 1.2 Рутокен Lite
  - 1.3 Рутокен ЭЦП 2.0
  - 1.4 Рутокен ЭЦП 3.0
- 2 Начало работы
  - 2.1 Подключение библиотеки
  - 2.2 Инициализация и деинициализация библиотеки
  - 2.3 Определение подключенных устройств
  - 2.4 Мониторинг событий в слоте
  - 2.5 Определение типа устройств
  - 2.6 Открытие и закрытие сессии
  - 2.7 Получение и сброс прав доступа
- 3 Управление объектами на токене
  - 3.1 Общие атрибуты объектов
  - 3.2 Создание объекта на токене
  - 3.3 Импорт объектов на токен
  - 3.4 Поиск объектов на токене
  - 3.5 Чтение и изменение объектов
  - 3.6 Удаление объектов на токене
- 4 Генерация ключевой пары
  - 4.1 Атрибуты ключевых объектов
  - 4.2 Поддерживаемые типы ключей
  - 4.3 Примеры шаблонов ключей ГОСТ Р 34.10-2012
  - 4.4 Поддерживаемые механизмы генерации ключей
  - 4.5 Пример генерации ключевой пары ГОСТ Р 34.10-2012
  - 4.6 Примеры шаблонов ключей ECDSA
  - 4.7 Пример генерации ключевой пары ECDSA
- 5 Генерация секретного ключа
  - 5.1 Атрибуты ключевых объектов
  - 5.2 Поддерживаемые типы ключей
  - 5.3 Примеры шаблона секретного ключа
  - 5.4 Поддерживаемые механизмы генерации ключей
  - 5.5 Пример генерации секретного ключа
- 6 Выработка сеансового симметричного ключа
  - 6.1 VKO GOST R 34.10-2001
  - 6.2 VKO GOST R 34.10-2012 (256 бит и 512 бит)
  - 6.3 KDF\_TREE\_GOSTR3411\_2012\_256
  - 6.4 SKM\_VENDOR\_GOST\_KEG
  - 6.5 Пример выработки общего ключа парной связи по алгоритму VKO GOST R 34.10-2012
  - 6.6 Пример выработки двойственного ключа по алгоритму Kef
  - 6.7 Маскирование секретного ключа
  - 6.8 Пример экспорта и импорта ключа по алгоритму Kexr15
- 7 Вычисление значения хеш-функции
  - 7.1 Поддерживаемые механизмы
  - 7.2 Хеширование данных
    - 7.2.1 Пример хеширования данных по алгоритму ГОСТ Р 34.11-94 и ГОСТ Р 34.11-2012
- 8 Вычисление MAC (Message Authentication Code)
  - 8.1 Поддерживаемые механизмы
  - 8.2 Вычисление MAC
    - 8.2.1 Пример получения MAC от данных по алгоритму ГОСТ Р 34.12-2015
  - 8.3 Проверка MAC
    - 8.3.1 Пример проверки MAC от данных по алгоритму ГОСТ Р 34.12-2015
- 9 Подпись и проверка подписи
  - 9.1 Поддерживаемые механизмы
  - 9.2 Подпись данных
    - 9.2.1 Подпись данных отдельными механизмами хеширования и подписи
    - 9.2.2 Пример подписи данных по алгоритму ГОСТ Р 34.10-2012 отдельными механизмами хеширования и подписи для всех устройств Рутокен
    - 9.2.3 Подпись совместным механизмом хеширования и подписи
    - 9.2.4 Пример подписи данных по алгоритму ГОСТ Р 34.10-2012 совместным механизмом хеширования и подписи
    - 9.2.5 Пример подписи данных по алгоритму ECDSA отдельными механизмами хеширования и подписи
  - 9.3 Поточная подпись и проверка подписи

- 10 Шифрование и расшифрование
  - 10.1 Поддерживаемые механизмы
  - 10.2 Шифрование данных
  - 10.3 Пример шифрования данных по алгоритмам ГОСТ 28147-89 и ГОСТ Р 34.12-2015
  - 10.4 Расшифрование данных
  - 10.5 Пример расшифрования данных по алгоритму ГОСТ 28147-89
- 11 Управление устройством
  - 11.1 Форматирование токена
  - 11.2 Установка и смена локального PIN-кода
  - 11.3 Разблокировка PIN-кода Пользователя
- 12 Управление памятью Рутокен ЭЦП Flash
  - 12.1 Получение объема флеш-памяти
  - 12.2 Создание разделов флеш-памяти
  - 12.3 Получение информации о разделах флеш-памяти
  - 12.4 Изменение атрибутов разделов флеш-памяти

## Введение

Функциональность устройств Рутокен охватывает широкий спектр задач обеспечения информационной безопасности. Устройства Рутокен могут быть применены для строгой двухфакторной аутентификации, защиты электронной переписки, установления защищенных соединений, безопасного проведения финансовых транзакций и криптографической защиты информации.

## Рутокен S

Рутокен S – это носитель ключевой информации и устройство для авторизации в компьютерных системах и защиты персональных данных с реализацией отечественного стандарта шифрования "на борту".

Через интерфейс PKCS#11 доступны следующие возможности Рутокен S:

- создание, запись, чтение, изменение, удаление двоичных файлов,
- генерация и импорт ключей шифрования ГОСТ 28147-89,
- шифрование данных по алгоритму ГОСТ 28147-89 в режимах простой замены, гаммирования и гаммирования с обратной связью,
- вычисление имитовставки по алгоритму ГОСТ 28147-89 длиной 32 бит,
- генерация последовательности случайных чисел длиной 256 бит.

Ограничения Rutoken S



- Рутокен S для работы требует установки драйверов: [для Windows](#), [для Linux](#), [для macOS](#).
- Рутокен S **не поддерживается** библиотекой **rtpkcs11esp**.
- Рутокен S в ОС Windows поддерживается устаревшей библиотекой **rtPKCS11.dll** и очень ограниченно поддерживается библиотекой **open sc-pkcs11** из состава [OpenSC](#).

## Рутокен Lite

Рутокен Lite – это ключевой носитель для безопасного хранения ключей шифрования и электронной подписи, паролей и других данных во встроенной защищенной памяти устройства.

Через интерфейс PKCS#11 доступны следующие возможности Рутокен Lite:

- создание, запись, чтение, изменение, удаление двоичных файлов.

## Рутокен ЭЦП 2.0

Рутокен ЭЦП 2.0 – электронный идентификатор с аппаратной реализацией отечественных и зарубежных стандартов электронной подписи, шифрования и хеширования.

Через интерфейс PKCS#11 доступны следующие возможности Рутокен ЭЦП 2.0:

- алгоритмы ГОСТ Р 34.10-2012 и ГОСТ Р 34.10-2001: генерация ключевых пар с проверкой качества, импорт ключевых пар, формирование и проверка электронной подписи,
- алгоритмы ГОСТ Р 34.11-2012 и ГОСТ Р 34.11-94: вычисление значения хеш-функции данных, в том числе с возможностью последующего формирования электронной подписи внутри устройства,
- алгоритмы ГОСТ 28147-89: генерация и импорт ключей шифрования, шифрование данных в режимах простой замены, гаммирования и гаммирования с обратной связью, вычисление и проверка криптографической контрольной суммы данных (имитовставки).

- выработка сессионных ключей (ключей парной связи): по схемам VKO GOST R 34.10-2012 (RFC 7836), VKO GOST R 34.10-2001 (RFC 4357), расшифрование по схеме EC El-Gamal.
- алгоритм RSA: поддержка ключей размером до 2048 бит, генерация ключевых пар с настраиваемой проверкой качества, импорт ключевых пар, формирование электронной подписи.
- генерация последовательности случайных чисел требуемой длины.

## Рутокен ЭЦП 3.0

Рутокен ЭЦП 3.0 – новый электронный идентификатор с аппаратной реализацией отечественных и зарубежных стандартов электронной подписи, шифрования и хеширования.

Через интерфейс PKCS#11 доступны следующие возможности Рутокен ЭЦП 3.0:

- алгоритмы ГОСТ Р 34.10-2012 и ГОСТ Р 34.10-2001: генерация ключевых пар с проверкой качества, импорт ключевых пар, формирование и проверка электронной подписи,
- алгоритмы ГОСТ Р 34.11-2012 и ГОСТ Р 34.11-94: вычисление значения хеш-функции данных, в том числе с возможностью последующего формирования электронной подписи внутри устройства,
- алгоритмы ГОСТ 28147-89 и ГОСТ Р 34.12-2015: генерация и импорт ключей шифрования, шифрование данных в режимах простой замены, гаммирования и гаммирования с обратной связью, вычисление и проверка криптографической контрольной суммы данных (имитовставки).
- выработка сессионных ключей (ключей парной связи): по схемам VKO GOST R 34.10-2012 (RFC 7836), VKO GOST R 34.10-2001 (RFC 4357), расшифрование по схеме EC El-Gamal.
- алгоритм RSA: поддержка ключей размером до 4096 бит, генерация ключевых пар с настраиваемой проверкой качества, импорт ключевых пар, формирование электронной подписи.
- алгоритм ECDSA с кривыми secp256k1 и secp256r1: генерация ключевых пар с настраиваемой проверкой качества, импорт ключевых пар, формирование электронной подписи.
- генерация последовательности случайных чисел требуемой длины.

## Начало работы

### Подключение библиотеки

Для работы с устройствами Рутокен через программный интерфейс PKCS#11 приложение должно предварительно загрузить библиотеку, содержащую реализацию функций и механизмов стандарта PKCS#11.

Рутокен SDK предоставляет две библиотеки `rtPKCS11` и `rtPKCS11ECP`, подробнее об особенностях выбора и использования которых можно ознакомиться в разделе [Использование библиотек `rtPKCS11` и `rtPKCS11ECP`](#). Основная разница заключается в том, что российские алгоритмы доступны в библиотеке `rtPKCS11ECP`, а зарубежные – в `rtPKCS11`.

Кроме функций стандартного интерфейса PKCS#11 библиотеки экспортируют функции расширения, которые могут быть удобны при использовании специфической функциональности устройств Рутокен.

После загрузки библиотеки нужно получить адрес экспортируемой библиотекой функции `C_GetFunctionList()` и вызвать ее для получения списка функций PKCS#11. Теперь все готово для работы с библиотекой.

Для работы с функциями расширения необходимо получить адрес функции `CK_C_EX_GetFunctionListExtended()` и вызвать ее для получения списка функций расширения PKCS#11.

После работы с библиотекой ее нужно выгрузить из памяти.

#### Загрузка библиотеки и списка функций PKCS#11

```

/* PKCS#11 */
#ifdef _WIN32
/* S, Lite, RSA */
#define PKCS11_LIBRARY_NAME "rtPKCS11.dll"
/* Lite, RSA */
#define PKCS11ECP_LIBRARY_NAME "rtPKCS11ECP.dll"
#endif
#ifdef __unix__
/* Lite, RSA */
#define PKCS11_LIBRARY_NAME "librtpkcs11ecp.so"
#define PKCS11ECP_LIBRARY_NAME "librtpkcs11ecp.so"
#endif
#ifdef __APPLE__
/* Lite, RSA */
#define PKCS11_LIBRARY_NAME "librtpkcs11ecp.dylib"
#define PKCS11ECP_LIBRARY_NAME "librtpkcs11ecp.dylib"
#endif

HMODULE hModule = NULL_PTR; // PKCS#11
CK_FUNCTION_LIST_PTR pFunctionList = NULL_PTR; // PKCS#11, CK_FUNCTION_LIST
CK_C_GetFunctionList pfGetFunctionList = NULL_PTR; // C_GetFunctionList
CK_RV rv = CKR_OK; //

while (TRUE)
{
    /* */
    printf("Loading library %s", PKCS11ECP_LIBRARY_NAME);
    hModule = LoadLibrary(PKCS11ECP_LIBRARY_NAME);
    if (hModule == NULL_PTR)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /* */
    printf("Getting GetFunctionList function");
    pfGetFunctionList = (CK_C_GetFunctionList)GetProcAddress(hModule, "C_GetFunctionList");
    if (pfGetFunctionList == NULL_PTR)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /* */
    printf("Getting function list");
    rv = pfGetFunctionList(&pFunctionList);
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
    ...
    break;
}

/* */
if (hModule)
{
    printf("Unloading library");
    if (FreeLibrary(hModule) != TRUE)
        printf(" -> Failed\n");
    else
        printf(" -> OK\n");
    hModule = NULL_PTR;
}

```

## Инициализация и деинициализация библиотеки

После загрузки библиотеки ее нужно инициализировать вызовом функции `C_Initialize()`. Параметр `NULL` при вызове данной функции означает, что функции библиотеки не будут вызываться из нескольких потоков, в противном случае в параметре должен быть передан указатель на структуру типа `CK_INITIALIZE_ARGS`.

Для завершения работы с библиотекой ее нужно деинициализировать вызовом функции `C_Finalize()`.

### Инициализация библиотеки

```
...

/* */
printf("Initializing library");
rv = pFunctionList->C_Initialize(NULL_PTR);
if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");

...

/* */
if (pFunctionList)
{
    printf("Finalizing library");
    rvTemp = pFunctionList->C_Finalize(NULL_PTR);
    if (rvTemp != CKR_OK)
        printf(" -> Failed\n");
    else
        printf(" -> OK\n");
    pFunctionList = NULL_PTR;
}
```

## Определение подключенных устройств

Доступ к каждому подключенному устройству осуществляется с помощью идентификатора слота, к которому оно подключено. Для получения списка всех слотов предназначена функция `C_GetSlotList()`. Значение первого параметра указывает, должен ли список включать слоты только с подключенными токенами (`CK_TRUE`) или все слоты (`CK_FALSE`).

## Получение списка токенов

```
CK_SLOT_ID_PTR aSlots = NULL_PTR;          //
CK_ULONG ulSlotCount = 0;                  //

while(TRUE)
{
...
    /* c */
    printf(" Getting number of connected slots");
    rv = pFunctionList->C_GetSlotList(CK_TRUE, NULL_PTR, &ulSlotCount);
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    aSlots = (CK_SLOT_ID*)malloc(ulSlotCount * sizeof(CK_SLOT_ID));
    if (aSlots == NULL)
    {
        printf("Memory allocation for aSlots failed! \n");
        break;
    }
    memset(aSlots, 0, (ulSlotCount * sizeof(CK_SLOT_ID)));

    /* c */
    printf(" Getting list of connected slots");
    rv = pFunctionList->C_GetSlotList(CK_TRUE, aSlots, &ulSlotCount);
    if (rv != CKR_OK)
    {
        printf(" -> Failed %X\n", (int)rv);
        break;
    }
    printf(" -> OK\n");

    printf(" Slots available: 0x%8.8X\n", (int)ulSlotCount);
    ...
    break;
}

if (aSlots)
{
    free(aSlots);
    aSlots = NULL_PTR;
}
}
```

Для получения актуальной информации о состоянии конкретного слота вызывается функция `C_GetSlotInfo()`, в которую передается идентификатор слота. Ее вызов запускает обновление информации обо всех слотах. Если токен извлечь из разъема и затем снова вставить в тот же самый разъем, то он может подключиться к любому свободному слоту, а не обязательно к тому же самому.

## Мониторинг событий в слоте

Для мониторинга событий извлечения и подключения токенов для всех слотов используется функция `C_WaitForSlotEvent()`, запущенная в отдельном потоке.

При вызове `C_WaitForSlotEvent()` с флагом `CKF_DONT_BLOCK` функция возвращает код `CKR_NO_EVENT` при отсутствии событий или код `CKR_OK` при его наличии (вместе с идентификатором соответствующего событию слота).

При вызове `C_WaitForSlotEvent()` с флагом `0` выполнение функции блокируется до возникновения события и функция возвращает код `CKR_OK` и номер соответствующего слота.

## Мониторинг событий в слотах

```
/* , - */
#define MONITORING_THREADS_NUMBER 1
```

```

/* , */
typedef struct _MONITORING_THREADS_PARAMS
{
    CK_FUNCTION_LIST_PTR m_pFunctionList;
    CK_FLAGS m_flags;
    DWORD m_dwThread_Number;
} MONITORING_THREADS_PARAMS, * PMONITORING_THREADS_PARAMS;

/* , .
*/
void Monitoring_Slots(IN void* param) // MONITORING_THREADS_PARAMS
{
    CK_FUNCTION_LIST_PTR pFunctionList = NULL_PTR; // PKCS#11, CK_FUNCTION_LIST
    CK_SLOT_ID slotID = 0xFFFFFFFF; // ,
    CK_SLOT_INFO slotInfo; // CK_SLOT_INFO
    CK_FLAGS ckFlags = 0; // , C_ cWaitForSlotEvent
    DWORD dwThreadNumber = 0; //
    CK_RV rv = CKR_OK; //

    /* MONITORING_THREADS_PARAMS */
    PMONITORING_THREADS_PARAMS pMonitoring_Threads_Param = (PMONITORING_THREADS_PARAMS)param;
    pFunctionList = pMonitoring_Threads_Param->m_pFunctionList;
    ckFlags = pMonitoring_Threads_Param->m_flags;
    dwThreadNumber = pMonitoring_Threads_Param->m_dwThread_Number;

    while (TRUE)
    {
        /* , C_WaitForSlotEvent ckFlags */
        slotID = 0xFFFFFFFF;
        rv = pFunctionList->C_WaitForSlotEvent(ckFlags, // 0 CKF_DONT_BLOCK
                                             &slotID, // ,
                                             NULL_PTR);

        if (rv == CKR_CRYPTOKI_NOT_INITIALIZED)
        {
            printf("Work with PKCS#11 has been finished.\n");
            break;
        }
        if (rv == CKR_NO_EVENT)
        {
            printf(" -> Failed \n"
                  "No more slot events...\n");
            break;
        }
        if (rv != CKR_OK)
        {
            printf(" -> Failed\n");
            break;
        }
        memset(&slotInfo, 0, sizeof(CK_SLOT_INFO));

        /* */
        rv = pFunctionList->C_GetSlotInfo(slotID,
                                         &slotInfo);

        if (rv != CKR_OK)
        {
            printf(" -> Failed\n");
            break;
        }

        /* */
        printf("\n Monitoring thread: 0x%8.8x \n", (int)dwThreadNumber);
        printf(" Slot ID: 0x%8.8x \n", (int)slotID);
        if (slotInfo.flags & CKF_TOKEN_PRESENT)
            printf(" Token has been attached!\n");
        else
            printf(" Token has been detached!\n");
    }
    printf("Exiting from thread: 0x%8.8x \n\n", (int)dwThreadNumber);
}

```

```

int main(int argc, char* argv[])
{
    DWORD i = 0; // .
    ...

    while (TRUE)
    {
        ...
        printf("\nPlease attach or detach Rutoken and press Enter...\n");
        getchar();
        i = 1;
        while (TRUE)
        {
            printf("Events counter: 0x%8.8x \n", (int)i);
            /* , */
            printf("C_WaitForSlotEvent");
            rv = pFunctionList->C_WaitForSlotEvent(CKF_DONT_BLOCK, // // ,
                                                    &slotID, // ,
                                                    NULL_PTR); // , NULL_PTR

            if (rv == CKR_NO_EVENT)
            {
                printf(" -> OK\n");
                printf("No more slots events.\n");
                break;
            }
            if (rv != CKR_OK)
            {
                printf(" -> Failed\n");
                break;
            }
            printf(" -> OK\n");
        }

        if ((rv != CKR_NO_EVENT) && (rv != CKR_OK))
            break;

        /* , - .
        .
        */
        while (TRUE)
        {
            MONITORING_THREADS_PARAMS aThreads_With_Blocking[MONITORING_THREADS_NUMBER];
            uintptr_t aThreads[MONITORING_THREADS_NUMBER];
            for (i = 0;
                i < MONITORING_THREADS_NUMBER;
                i++)
            {
                printf("Starting monitoring thread number 0x%8.8X \n", (int)i);
                memset(&aThreads_With_Blocking[i],
                    0,
                    sizeof(MONITORING_THREADS_PARAMS));
                aThreads_With_Blocking[i].m_pFunctionList = pFunctionList;
                aThreads_With_Blocking[i].m_flags = 0;
                aThreads_With_Blocking[i].m_dwThread_Number = i;
                aThreads[i] = CreateProc(&aThreads[i], NULL_PTR, &Monitoring_Slots,
&aThreads_With_Blocking[i]);
            }
            printf("\n\nPlease attach or detach Rutoken or press Enter to exit.\n");
            getchar();
            break;
        }
        break;
    }
}

```

## Определение типа устройств



Стандарт PKCS#11 предлагает определять тип устройства по параметрам, возвращаемым функциями `C_GetSlotInfo()` и `C_GetTokenInfo()`. Функция `C_GetSlotInfo()` возвращает структуру типа `CK_SLOT_INFO`, содержащую в поле `slotDescription` имя считывателя и в поле `manufacturerID` производителя устройства (Aktiv Co.). Функция `C_GetTokenInfo()` возвращает структуру типа `CK_TOKEN_INFO`, содержащую в поле `model` наименование модели устройства. Имя считывателя или наименование устройства обычно позволяет однозначно идентифицировать тип подключенного устройства. В таблице приведены значения для каждого из устройств Рутокен.

Модель Рутокен	Значение <code>slotDescription</code> структуры <code>CK_SLOT_INFO</code>	Значение <code>model</code> структуры <code>CK_TOKEN_INFO</code>
Рутокен S	Aktiv Co. ruToken 0	Rutoken S 64K, где 64K - размер памяти
Рутокен Lite	Aktiv Rutoken lite 0	Rutoken lite
Рутокен ЭЦП	Aktiv Rutoken ECP 0	Rutoken ECP

Более удобным способом получить информацию о подключенном к слоту токене можно с помощью функции расширения `C_EX_GetTokenInfoExtended()`, которая возвращает расширенные данные в виде структуры типа `CK_TOKEN_INFO_EXTENDED`. Поля `ulTokenClass` и `ulTokenType` содержат информацию о классе и типе устройства соответственно и могут быть использованы для их определения.

## Определение класса токена

```
CK_C_EX_GetFunctionListExtended pfGetFunctionListEx = NULL_PTR;          // C_EX_GetFunctionListExtended
CK_FUNCTION_LIST_EXTENDED_PTR   pFunctionListEx = NULL_PTR;            // PKCS#11,
CK_FUNCTION_LIST_EXTENDED
CK_TOKEN_INFO_EXTENDED          tokenInfoEx;                          // CK_TOKEN_INFO_EXTENDED

while(TRUE)
{
    ...

    printf("Determining token type");

    /*          */
    pfGetFunctionListEx = (CK_C_EX_GetFunctionListExtended)GetProcAddress(hModule,

"C_EX_GetFunctionListExtended");
    if (pfGetFunctionListEx == NULL_PTR)
    {
        printf(" -> Failed\n");
        break;
    }

    /*          */
    rv = pfGetFunctionListEx(&pFunctionListEx);
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    memset(&tokenInfoEx,
          0,
          sizeof(CK_TOKEN_INFO_EXTENDED));
    tokenInfoEx.ulSizeOfThisStructure = sizeof(CK_TOKEN_INFO_EXTENDED);
    /*          */
    rv = pFunctionListEx->C_EX_GetTokenInfoExtended(aSlots[0],          // ,
                                                    &tokenInfoEx);      //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    /*          */
    switch (tokenInfoEx.ulTokenClass)
    {
        case TOKEN_CLASS_S:
            printf(": Rutoken / Rutoken S\n");
        case TOKEN_CLASS_ECP:
            printf(": Rutoken ECP\n");
        case TOKEN_CLASS_LITE:
            printf(": Rutoken Lite\n");
        default:
            printf(": undefined\n");
    }

    break;
}
}
```

## Открытие и закрытие сессии

Большинство функций PKCS#11 требует наличие открытой сессии между токеном и приложением.

Для открытия сессии используется функция `C_OpenSession()`, для закрытия сессии – `C_CloseSession()`, для закрытия всех открытых сессий – `C_CloseAllSessions()`.

Сессия может быть открыта только для чтения объектов на токене или для чтения и записи (флаг CKF\_RW\_SESSION). После открытия сессии приложение получает доступ к публичным объектам на токене. Для доступа к приватным объектам пользователь должен получить доступ Пользователя или Администратора функций C\_Login().

При закрытии сессии все сессионные объекты уничтожаются, даже если приложение использует эти объекты в других сессиях.

### Мониторинг событий в слотах

```
CK_SESSION_HANDLE hSession = NULL_PTR;      //

...

/* RW      */
printf("Opening Session");
rv = pFunctionList->C_OpenSession(aSlots[0],                                     //

CKF_RW_SESSION,                          //

CKF_SERIAL_SESSION |

NULL_PTR,

NULL_PTR,

&hSession);                               //
if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");

...

/*      */
printf("C_CloseAllSession");
rv = pFunctionList->C_CloseAllSessions(aSlots[0]);
if (rvTemp != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
hSession = NULL_PTR;
```

### Получение и сброс прав доступа

В PKCS#11 доступны две глобальные роли: CKU\_USER – пользователь Рутокен, CKU\_SO – администратор Рутокен. Помимо них, устройства Рутокен ЭЦП имеют локальные роли, про которые можно ознакомиться в разделе [Установка и смена локального PIN-кода](#).

Для аутентификации предварительно необходимо открыть сессию.

## Мониторинг событий в слотах

```
/* DEMO PIN- */
CK_UTF8CHAR USER_PIN[] = {'1', '2', '3', '4', '5', '6', '7', '8'};

...

/* */
printf("Logging in");
rv = pFunctionList->C_Login(hSession,
                             CKU_USER,
                             USER_PIN,
                             sizeof(USER_PIN));
//
// PIN-

if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");

...

/* */
printf("Logging out");
rv = pFunctionList->C_Logout(hSession);
if ((rv == CKR_OK) || (rv == CKR_USER_NOT_LOGGED_IN))
    printf(" -> OK\n");
else
    printf(" -> Failed\n");
```

## Управление объектами на токене

### Общие атрибуты объектов

Стандарт PKCS#11 различает несколько классов объектов. Объекты содержат набор атрибутов, каждый из которых имеет только одно определенное значение.

Атрибут `CKA_TOKEN` определяет, будет ли созданный объект храниться только в рамках текущей сессии (значение `CK_FALSE`), или будет сохранен в памяти Рутокен (значение `CK_TRUE`). Сессионный объект будет уничтожен автоматически при закрытии сессии, а сохраненный на токене доступен вплоть до физического удаления объекта.

Атрибут `CKA_PRIVATE` определяет доступность объекта. Пользователь не имеет доступа к приватному объекту (значение `CK_TRUE`) до тех пор, пока не выполнит аутентификацию на токене. Публичный же объект доступен без аутентификации (значение `CK_FALSE`).

Атрибут `CKA_MODIFIABLE` отвечает за возможность изменения атрибутов объекта после его создания. По умолчанию атрибут принимает значение `CK_TRUE`, при котором редактирование атрибутов объекта становится возможным. Значение `CK_FALSE` означает, что созданный объект будет доступен «только для чтения» и значения атрибутов объекта после его создания не могут быть изменены.

Сгенерированные устройствами Рутокен закрытые и секретные ключевые объекты в целях безопасности является неизвлекаемыми. Это означает, что значение ключа (значение атрибута `CKA_VALUE`) невозможно получить через функцию `C_GetAttributeValue()`. Все криптографические операции с такими ключами производятся внутри устройства без передачи значения ключа наружу.

### Создание объекта на токене

Для создания объектов на токене предназначена функция `C_CreateObject()`. В нее передается предварительно созданный шаблон с атрибутами создаваемого объекта, размер шаблона и хэндл открытой сессии с правами Пользователя. Функция возвращает хэндл созданного с указанными атрибутами объекта.

Устройства Рутокен, сертифицированные ФСБ, не поддерживают создание (импорт) ключей функцией `C_CreateObject` по алгоритмам ГОСТ 28147-89, ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012 в долговременную память (с флагом `CKA_TOKEN = TRUE`).

## Создание объекта типа CKO\_DATA на токене

```
CK_OBJECT_CLASS ocData = CKO_DATA;
CK_UTF8CHAR      label[] = "Data object sample";
CK_UTF8CHAR      application[] = "Rutoken Control Panel";
CK_BYTE          data[] = "Sample data";
CK_BBOOL         true = CK_TRUE;

CK_ATTRIBUTE attrDataTmpl[] = {
    {CKA_CLASS, &ocData, sizeof(ocData)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_VALUE, data, sizeof(data)},
    {CKA_APPLICATION, application, sizeof(application)-1},
    {CKA_TOKEN, &>true, sizeof(true)}
};

CK_OBJECT_HANDLE hData; //      CKO_DATA

...

printf("Create object");
rv = pFunctionList->C_CreateObject( hSession,                                //
                                     attrDataTmpl,
//
                                     arraysize(attrDataTmpl), //
                                     &hData);

//
if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
```

## Импорт объектов на токен

Для импорта объектов также используется функция `C_CreateObject()`. В нее передается предварительно созданный шаблон с атрибутами импортируемого объекта (в том числе и значением `CKA_VALUE`), размер шаблона и хэндл открытой сессии с правами Пользователя. Функция возвращает хэндл созданного с указанными атрибутами объекта.

Устройства Рутокен, сертифицированные ФСБ, не поддерживают создание (импорт) ключей функцией `C_CreateObject` по алгоритмам ГОСТ 28147-89, ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012 в долговременную память (с флагом `CKA_TOKEN = TRUE`).

## Импорт сертификата

```
/*      */
CK_BYTE cbCertificate[] = { 0x30, 0x82, 0x03, 0x36, 0x30, 0x82, 0x02, 0xe5,
                            0xa0, 0x03, 0x02, 0x01, 0x02, 0x02, 0x0a, 0x24,
                            0xa5, 0x24, 0x63, 0x00, 0x02, 0x00, 0x01, 0xa7,
                            0x15, 0x30, 0x08, 0x06, 0x06, 0x2a, 0x85, 0x03,
                            0x02, 0x02, 0x03, 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, 0x6e, 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, 0x4e, 0x2d, 0x50,
                            0x52, 0x4f, 0x30, 0x1e, 0x17, 0x0d, 0x31, 0x31,
                            0x31, 0x31, 0x32, 0x32, 0x31, 0x30, 0x31, 0x33,
                            0x34, 0x32, 0x5a, 0x17, 0x0d, 0x31, 0x34, 0x31,
                            0x30, 0x30, 0x34, 0x30, 0x37, 0x30, 0x39, 0x34,
```

0x31, 0x5a, 0x30, 0x65, 0x31, 0x10, 0x30, 0x0e,  
0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x07, 0x49,  
0x76, 0x61, 0x6e, 0x6f, 0x66, 0x66, 0x31, 0x0b,  
0x30, 0x09, 0x06, 0x03, 0x55, 0x04, 0x06, 0x13,  
0x02, 0x52, 0x55, 0x31, 0x14, 0x30, 0x12, 0x06,  
0x03, 0x55, 0x04, 0x05, 0x13, 0x0b, 0x31, 0x32,  
0x33, 0x31, 0x32, 0x33, 0x31, 0x32, 0x33, 0x31,  
0x32, 0x31, 0x1d, 0x30, 0x1b, 0x06, 0x09, 0x2a,  
0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x09, 0x01,  
0x16, 0x0e, 0x69, 0x76, 0x61, 0x6e, 0x6f, 0x76,  
0x40, 0x6d, 0x61, 0x69, 0x6c, 0x2e, 0x72, 0x75,  
0x31, 0x0f, 0x30, 0x0d, 0x06, 0x03, 0x55, 0x04,  
0x08, 0x13, 0x06, 0x4d, 0x6f, 0x73, 0x63, 0x6f,  
0x77, 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, 0xfb, 0x3c,  
0xdc, 0x59, 0xc3, 0x9c, 0x4a, 0x43, 0x89, 0x87,  
0xc7, 0xd7, 0xfe, 0x50, 0x19, 0xb3, 0x0c, 0x8b,  
0x76, 0x97, 0xa9, 0xdf, 0xb7, 0xca, 0x2c, 0x6c,  
0x3b, 0xa9, 0x13, 0xf4, 0xe0, 0x69, 0x02, 0x59,  
0x92, 0x47, 0x21, 0x1a, 0xef, 0x90, 0x61, 0x91,  
0x40, 0x30, 0xdd, 0x7c, 0xb0, 0x4f, 0x64, 0x5f,  
0x24, 0x9a, 0xf1, 0xd6, 0x0f, 0xa9, 0xf0, 0x86,  
0xd9, 0x35, 0x2b, 0x3e, 0xf2, 0xf3, 0xa3, 0x82,  
0x01, 0x73, 0x30, 0x82, 0x01, 0x6f, 0x30, 0x0b,  
0x06, 0x03, 0x55, 0x1d, 0x0f, 0x04, 0x04, 0x03,  
0x02, 0x04, 0xf0, 0x30, 0x26, 0x06, 0x03, 0x55,  
0x1d, 0x25, 0x04, 0x1f, 0x30, 0x1d, 0x06, 0x07,  
0x2a, 0x85, 0x03, 0x02, 0x02, 0x22, 0x06, 0x06,  
0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x03,  
0x02, 0x06, 0x08, 0x2b, 0x06, 0x01, 0x05, 0x05,  
0x07, 0x03, 0x04, 0x30, 0x1d, 0x06, 0x03, 0x55,  
0x1d, 0x0e, 0x04, 0x16, 0x04, 0x14, 0x8a, 0x24,  
0x35, 0x74, 0x6b, 0xf7, 0x91, 0x17, 0x92, 0xb2,  
0xcf, 0x8f, 0x63, 0x87, 0xb7, 0x69, 0x06, 0xe1,  
0x71, 0xf2, 0x30, 0x1f, 0x06, 0x03, 0x55, 0x1d,  
0x23, 0x04, 0x18, 0x30, 0x16, 0x80, 0x14, 0x6d,  
0x8f, 0x5e, 0x05, 0xd9, 0x5f, 0xac, 0x91, 0x17,  
0x94, 0x1e, 0x95, 0x9a, 0x05, 0x30, 0x38, 0x37,  
0x7a, 0x10, 0x2a, 0x30, 0x55, 0x06, 0x03, 0x55,  
0x1d, 0x1f, 0x04, 0x4e, 0x30, 0x4c, 0x30, 0x4a,  
0xa0, 0x48, 0xa0, 0x46, 0x86, 0x44, 0x68, 0x74,  
0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77, 0x77, 0x77,  
0x2e, 0x63, 0x72, 0x79, 0x70, 0x74, 0x6f, 0x70,  
0x72, 0x6f, 0x2e, 0x72, 0x75, 0x2f, 0x43, 0x65,  
0x72, 0x74, 0x45, 0x6e, 0x72, 0x6f, 0x6c, 0x6c,  
0x2f, 0x54, 0x65, 0x73, 0x74, 0x25, 0x32, 0x30,  
0x43, 0x65, 0x6e, 0x74, 0x65, 0x72, 0x25, 0x32,  
0x30, 0x43, 0x52, 0x59, 0x50, 0x54, 0x4f, 0x2d,  
0x50, 0x52, 0x4f, 0x28, 0x32, 0x29, 0x2e, 0x63,  
0x72, 0x6c, 0x30, 0x81, 0xa0, 0x06, 0x08, 0x2b,  
0x06, 0x01, 0x05, 0x05, 0x07, 0x01, 0x01, 0x04,  
0x81, 0x93, 0x30, 0x81, 0x90, 0x30, 0x33, 0x06,  
0x08, 0x2b, 0x06, 0x01, 0x05, 0x05, 0x07, 0x30,  
0x01, 0x86, 0x27, 0x68, 0x74, 0x74, 0x70, 0x3a,  
0x2f, 0x2f, 0x77, 0x77, 0x77, 0x2e, 0x63, 0x72,  
0x79, 0x70, 0x74, 0x6f, 0x70, 0x72, 0x6f, 0x2e,  
0x72, 0x75, 0x2f, 0x6f, 0x63, 0x73, 0x70, 0x6e,  
0x63, 0x2f, 0x6f, 0x63, 0x73, 0x70, 0x2e, 0x73,  
0x72, 0x66, 0x30, 0x59, 0x06, 0x08, 0x2b, 0x06,  
0x01, 0x05, 0x05, 0x07, 0x30, 0x02, 0x86, 0x4d,  
0x68, 0x74, 0x74, 0x70, 0x3a, 0x2f, 0x2f, 0x77,  
0x77, 0x77, 0x2e, 0x63, 0x72, 0x79, 0x70, 0x74,  
0x6f, 0x70, 0x72, 0x6f, 0x2e, 0x72, 0x75, 0x2f,  
0x43, 0x65, 0x72, 0x74, 0x45, 0x6e, 0x72, 0x6f,  
0x6c, 0x6c, 0x2f, 0x70, 0x6b, 0x69, 0x2d, 0x73,  
0x69, 0x74, 0x65, 0x5f, 0x54, 0x65, 0x73, 0x74,  
0x25, 0x32, 0x30, 0x43, 0x65, 0x6e, 0x74, 0x65,  
0x72, 0x25, 0x32, 0x30, 0x43, 0x52, 0x59, 0x50,



```

                                        arraysize(attrGOST28147SecKey));    //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

/*      */
printf("C_FindObjects");
//      100
phObject = (CK_OBJECT_HANDLE*)malloc(100 * sizeof(CK_OBJECT_HANDLE));
if (phObject == NULL)
{
    printf("Memory allocation for aSlots failed! \n");
    break;
}
memset(phObject,
        0,
        (100 * sizeof(CK_OBJECT_HANDLE)));
rv = pFunctionList->C_FindObjects(hSession,                                //
                                  phObject,                                //
                                  100,                                    //
                                  pulObjectCount);                        //

if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
break;
}

/*      */
printf("C_FindObjectsFinal");
rvTemp = pFunctionList->C_FindObjectsFinal(hSession);                    //
if (rvTemp != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
if (rv == CKR_OK)
    printf("Search has been completed.\n"
           "Objects found: %d \n",
           (int)ulObjectCount);
else
{
    printf("Search failed!\n");
    if (phObject)
    {
        free(phObject);
        phObject = NULL_PTR;
        ulObjectCount = 0;
    }
}
break;
}
}

```

## Чтение и изменение объектов

Для чтения атрибутов созданного объекта используется функция `C_GetAttributeValue()`. Если объекты или его атрибуты не являются защищенными от записи (например, значение атрибута `CKA_MODIFIABLE` равно `CK_TRUE`), то значения атрибутов могут быть изменены функцией `C_SetAttributeValue()`.

Сгенерированные устройства Рутокен закрытые и секретные ключевые объекты в целях безопасности является неизвлекаемыми. Это означает, что значение ключа (значение атрибута `CKA_VALUE`) невозможно получить через функцию `C_GetAttributeValue()`. Все криптографические операции с такими ключами производятся внутри устройства без извлечения ключа и передачи его значения наружу.

### Чтение и изменение атрибутов объекта

```
CK_BYTE          new_data[] = "Sample Rutoken data";
```



```

CK_ATTRIBUTE attrDataReadTmpl[] = {CKA_VALUE, NULL_PTR, NULL};
CK_ATTRIBUTE attrDataEditTmpl[] = {CKA_VALUE, new_data, sizeof(new_data)};

while(true)
{
    ...

    /* */
    printf("Setting object value");
    rv = pFunctionList->C_SetAttributeValue(hSession,          //
                                           hData,              //
                                           &attrDataEditTmpl, //
                                           1);                 //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      CKA_VALUE*/
    printf("Getting object value size");
    rv = pFunctionList->C_GetAttributeValue(hSession,          //
                                           hData,              //
                                           &attrDataReadTmpl, //
                                           1);                 //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      */
    attrDataReadTmpl.pValue = (CK_BYTE*)malloc(attrDataReadTmpl.ulValueLen);
    if (attrDataReadTmpl.pValue == NULL)
    {
        printf("Memory allocation for attrDataReadTmpl failed! \n");
        break;
    }
    memset(attrDataReadTmpl.pValue,
           0,
           (attrDataReadTmpl.ulValueLen * sizeof(CK_BYTE)));

    /* */
    printf("Getting object value");
    rv = pFunctionList->C_GetAttributeValue(hSession,          //
                                           hData,              //
                                           &attrDataReadTmpl, //
                                           1);                 //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      */
    printf("Data is:\n");
    for (i = 0;
         i < attrDataReadTmpl.ulValueLen;
         i++)
        printf("%s", attrDataReadTmpl.pValue[i]);
    break;
}

if (attrDataReadTmpl.pValue)
{
    free(attrDataReadTmpl.pValue);
}

```

```
attrDataReadTpl.pValue = NULL_PTR;
attrDataReadTpl.ulValueLen= 0;
}
```

## Удаление объектов на токене

Для удаления объекта на токене необходимо знать его хэндл, который следует передать в функцию `C_DestroyObject()` (предварительно должна быть открыта сессия с правами Пользователя). Если хэндл объекта неизвестен, то по любому известному атрибуту объекта можно получить его хэндл через группу функций `C_FindObjectsInit()`, `C_FindObjects()` `C_FindObjectsFinal()`.

### Удаление объектов на токене

```
/*      */
for (i = 0;
     i < ulObjectCount;
     i++)
{
    printf(" C_DestroyObject %d", (int)(i + 1));
    rv = pFunctionList->C_DestroyObject(hSession,          //
                                       phObject[i]);      //
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
}
if (ulObjectCount != 0)
    printf("Destruction objects has been completed successfully.\n");
free(phObject);
phObject = NULL_PTR;
ulObjectCount = 0;
```

## Генерация ключевой пары

### Атрибуты ключевых объектов

Все поддерживаемые устройства Рутокен атрибуты объектов представлены в разделе [Объекты PKCS #11](#).

### Поддерживаемые типы ключей

Устройства Рутокен поддерживают следующие типы ключей асимметричной криптографии (`CK_KEY_TYPE`):

- `CKK_GOSTR3410` для ключей ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012 (256 бит),
- `CKK_GOSTR3410_512` для ключей ГОСТ Р 34.10-2012 (512 бит),
- `CKK_RSA` для ключей RSA,
- `CKK_EC` для ключей ECDSA.

## Примеры шаблонов ключей ГОСТ Р 34.10-2012

Ниже представлены примеры шаблонов закрытого и открытого ключа ГОСТ Р 34.10-2012 с пояснениями.

## Шаблон закрытого ключа ГОСТ Р 34.10-2012

```
CK_OBJECT_CLASS      ocPrivKey = CKO_PRIVATE_KEY;
CK_UTF8CHAR          PrivKeyLabel[] = { "GOST Private Key" };
CK_BYTE              KeyPairID[] = { "GOST keypair" };
CK_KEY_TYPE          keyTypeGostR3410 = CKK_GOSTR3410;           // 256
CK_KEY_TYPE          keyTypeGostR3410_512 = CKK_GOSTR3410_512; // 512
CK_BBOOL              bTrue = CK_TRUE;

/* A 34.10-2012(256) */
CK_BYTE              parametersGostR3410_2012_256[] = { 0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01
};

/* A 34.10-2012(512) */
CK_BYTE              parametersGostR3410_2012_512[] = { 0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02,
0x01, 0x02, 0x01 };

/* 34.11-2012(256) */
CK_BYTE              parametersGostR3411_2012_256[] = { 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01,
0x02, 0x02 };

/* 34.11-2012(512) */
CK_BYTE              parametersGostR3411_2012_512[] = { 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01,
0x02, 0x03 };

CK_ATTRIBUTE GOST34_10_2012_256_PrivateKey[] = {
    { CKA_CLASS, &ocPrivKey, sizeof(ocPrivKey)}, // 256
    { CKA_LABEL, &PrivKeyLabel, sizeof(PrivKeyLabel) - 1}, //
    { CKA_ID, &KeyPairID, sizeof(KeyPairID) - 1}, // #1 (
    { CKA_KEY_TYPE, &keyTypeGostR3410, sizeof(keyTypeGostR3410)}, //
    { CKA_TOKEN, &bTrue, sizeof(bTrue)}, //
    { CKA_PRIVATE, &bTrue, sizeof(bTrue)}, //
    { CKA_DERIVE, &bTrue, sizeof(bTrue)}, // (
)
    { CKA_GOSTR3410_PARAMS, parametersGostR3410_2012_256, sizeof(parametersGostR3410_2012_256)},//
    { CKA_GOSTR3411_PARAMS, parametersGostR3411_2012_256, sizeof(parametersGostR3411_2012_256)} //
34.11-2012(256)
};

CK_ATTRIBUTE GOST34_10_2012_512_PrivateKey[] = {
    { CKA_CLASS, &ocPrivKey, sizeof(ocPrivKey)}, // 512
    { CKA_LABEL, &PrivKeyLabel, sizeof(PrivKeyLabel) - 1}, //
    { CKA_ID, &KeyPairID, sizeof(KeyPairID) - 1}, // #1 (
    { CKA_KEY_TYPE, &keyTypeGostR3410_512, sizeof(keyTypeGostR3410_512)}, //
    { CKA_TOKEN, &bTrue, sizeof(bTrue)}, //
    { CKA_PRIVATE, &bTrue, sizeof(bTrue)}, //
    { CKA_DERIVE, &bTrue, sizeof(bTrue)}, // (
)
    { CKA_GOSTR3410_PARAMS, parametersGostR3410_2012_512, sizeof(parametersGostR3410_2012_512)},//
    { CKA_GOSTR3411_PARAMS, parametersGostR3411_2012_512, sizeof(parametersGostR3411_2012_512)} //
34.11-2012(512)
};
```

## Шаблон открытого ключа ГОСТ Р 34.10-2012

```
CK_OBJECT_CLASS ocPubKey = CKO_PUBLIC_KEY;
CK_UTF8CHAR      PubKeyLabel[] = { "GOST Public Key" };
CK_BYTE          KeyPairID[] = { "GOST keypair" };
CK_KEY_TYPE      keyTypeGostR3410 = CKK_GOSTR3410;           // 256
CK_KEY_TYPE      keyTypeGostR3410_512 = CKK_GOSTR3410_512;   // 512
CK_BBOOL         bTrue = CK_TRUE;
CK_BBOOL         bFalse = CK_FALSE;

/* A 34.10-2012(256) */
CK_BYTE          parametersGostR3410_2012_256[] = { 0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01
};

/* A 34.10-2012(512) */
CK_BYTE          parametersGostR3410_2012_512[] = { 0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02,
0x01, 0x02, 0x01 };

/* 34.11-2012(256) */
CK_BYTE          parametersGostR3411_2012_256[] = { 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01,
0x02, 0x02 };

/* 34.11-2012(512) */
CK_BYTE          parametersGostR3411_2012_512[] = { 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01,
0x02, 0x03 };

CK_ATTRIBUTE      GOST34_10_2012_256_PublicKey[] = {           // , 256
    { CKA_CLASS, &ocPubKey, sizeof(ocPubKey)},                //
    { CKA_LABEL, &PubKeyLabel, sizeof(PubKeyLabel) - 1},      //
    { CKA_ID, &KeyPairID, sizeof(KeyPairID) - 1},             //
    { CKA_KEY_TYPE, &keyTypeGostR3410, sizeof(keyTypeGostR3410)}, //
    { CKA_TOKEN, &bTrue, sizeof(bTrue)},                      //
    { CKA_PRIVATE, &bFalse, sizeof(bFalse)},                  //
    { CKA_DERIVE, &bTrue, sizeof(bTrue)},                      // (
)
    { CKA_GOSTR3410_PARAMS, parametersGostR3410_2012_256, sizeof(parametersGostR3410_2012_256)}, //
    { CKA_GOSTR3411_PARAMS, parametersGostR3411_2012_256, sizeof(parametersGostR3411_2012_256)} //
};

CK_ATTRIBUTE      GOST34_10_2012_512_PublicKey[] = {           // , 512
    { CKA_CLASS, &ocPubKey, sizeof(ocPubKey)},                //
    { CKA_LABEL, &PubKeyLabel, sizeof(PubKeyLabel) - 1},      //
    { CKA_ID, &KeyPairID, sizeof(KeyPairID) - 1},             //
    { CKA_KEY_TYPE, &keyTypeGostR3410_512, sizeof(keyTypeGostR3410_512)}, //
    { CKA_TOKEN, &bTrue, sizeof(bTrue)},                      //
    { CKA_PRIVATE, &bFalse, sizeof(bFalse)},                  //
    { CKA_DERIVE, &bTrue, sizeof(bTrue)},                      // (
)
    { CKA_GOSTR3410_PARAMS, parametersGostR3410_2012_512, sizeof(parametersGostR3410_2012_512)}, //
    { CKA_GOSTR3411_PARAMS, parametersGostR3411_2012_512, sizeof(parametersGostR3411_2012_512)} //
34.11-2012(512)
};
```

## Поддерживаемые механизмы генерации ключей

Устройства Рутокен поддерживают следующие механизмы генерации ключевой пары:

- CKM\_GOSTR3410\_KEY\_PAIR\_GEN для генерации ключевой пары ГОСТ Р 34.10.2001 и ГОСТ Р 34.10.2012 с длиной ключа 256 бит,
- CKM\_GOSTR3410\_512\_KEY\_PAIR\_GEN для генерации ключевой пары ГОСТ Р 34.10.2012 с длиной ключа 512 бит,
- CKM\_RSA\_PKCS\_KEY\_PAIR\_GEN для генерации ключевой пары RSA,
- CKM\_EC\_KEY\_PAIR\_GEN для генерации ключевой пары ECDSA.

## Пример генерации ключевой пары ГОСТ Р 34.10-2012

Предварительно должна быть открыта сессия чтения/записи с авторизацией с правами пользователя Рутокен.

### Генерация ключевой пары ГОСТ Р 34.10-2012

```
/* */
#define          arraysize(a)  (sizeof(a)/sizeof(a[0]))

CK_MECHANISM    gostR3410_256KeyPairGenMech = { CKM_GOSTR3410_KEY_PAIR_GEN, NULL_PTR, 0 };           //
              34.10-2012(256)
CK_MECHANISM    gostR3410_512KeyPairGenMech = { CKM_GOSTR3410_512_KEY_PAIR_GEN, NULL_PTR, 0 };         //
              34.10-2012(512) */

CK_OBJECT_HANDLE hPublicKey_256 = NULL_PTR;           //
CK_OBJECT_HANDLE hPrivateKey_256 = NULL_PTR;         //
CK_OBJECT_HANDLE hPublicKey_512 = NULL_PTR;         //
CK_OBJECT_HANDLE hPrivateKey_512 = NULL_PTR;         //

...

printf("Generating 256 bit key pair");
rv = pFunctionList->C_GenerateKeyPair(hSession,          //
    &gostR3410_256KeyPairGenMech,                        //
    GOST34_10_2012_256_PublicKey,                       //
    arraysize(GOST34_10_2012_256_PublicKey),           //
    GOST34_10_2012_256_PrivateKey,                     //
    arraysize(GOST34_10_2012_256_PrivateKey),         //
    &hPublicKey_256,                                   //
    &hPrivateKey_256);                                //

if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");

printf("Generating 512 bit key pair");
rv = pFunctionList->C_GenerateKeyPair(hSession,          //
    &gostR3410_512KeyPairGenMech,                        //
    GOST34_10_2012_512_PublicKey,                       //
    arraysize(GOST34_10_2012_512_PublicKey),           //
    GOST34_10_2012_512_PrivateKey,                     //
    arraysize(GOST34_10_2012_512_PrivateKey),         //
    &hPublicKey_512,                                   //
    &hPrivateKey_512);                                //

if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
```

### Примеры шаблонов ключей ECDSA

Ниже представлены примеры шаблонов закрытого и открытого ключа ECDSA ключа

## Шаблоны открытого и закрытого ключа ECDSA

```
CK_OBJECT_CLASS publicKeyObject = CKO_PUBLIC_KEY;
CK_OBJECT_CLASS privateKeyObject = CKO_PRIVATE_KEY;

/* ECDSA */
CK_UTF8CHAR publicKeyLabelEcdsa[] = { "Sample ECDSA Public Key (Aktiv Co.)" };

/* ECDSA */
CK_UTF8CHAR privateKeyLabelEcdsa[] = { "Sample ECDSA Private Key (Aktiv Co.)" };

/* ID ECDSA */
CK_BYTE keyPairIdEcdsa[] = { "ECDSA sample key pair ID (Aktiv Co.)" };

CK_KEY_TYPE keyTypeEcdsa = CKK_EC;
CK_BBOOL attributeTrue = CK_TRUE;
CK_BBOOL attributeFalse = CK_FALSE;

// EC
CK_BYTE secp256k1Oid[] = { 0x06, 0x05, 0x2B, 0x81, 0x04, 0x00, 0x0A };

/*****
 * ECDSA *
*****/
CK_ATTRIBUTE publicKeyTemplate[] =
{
    { CKA_CLASS, &publicKeyObject, sizeof(publicKeyObject)}, // -
    { CKA_LABEL, &publicKeyLabelEcdsa, sizeof(publicKeyLabelEcdsa) - 1 }, //
    { CKA_ID, &keyPairIdEcdsa, sizeof(keyPairIdEcdsa) - 1 }, // ( )
    { CKA_KEY_TYPE, &keyTypeEcdsa, sizeof(keyTypeEcdsa) }, // - ECDSA
    { CKA_TOKEN, &attributeTrue, sizeof(attributeTrue)}, //
    { CKA_PRIVATE, &attributeFalse, sizeof(attributeFalse)}, //
    { CKA_EC_PARAMS, secp256k1Oid, sizeof(secp256k1Oid) } //
};

/*****
 * ECDSA *
*****/
CK_ATTRIBUTE privateKeyTemplate[] =
{
    { CKA_CLASS, &privateKeyObject, sizeof(privateKeyObject)}, // -
    { CKA_LABEL, &privateKeyLabelEcdsa, sizeof(privateKeyLabelEcdsa) - 1 }, //
    { CKA_ID, &keyPairIdEcdsa, sizeof(keyPairIdEcdsa) - 1 }, // #1 ( )
    { CKA_KEY_TYPE, &keyTypeEcdsa, sizeof(keyTypeEcdsa) }, // - ECDSA
    { CKA_TOKEN, &attributeTrue, sizeof(attributeTrue)}, //
    { CKA_PRIVATE, &attributeTrue, sizeof(attributeTrue)} //
};
```

## Пример генерации ключевой пары ECDSA

## Генерация ключевой пары ECDSA

```
/* */
#define          arraysize(a)  (sizeof(a)/sizeof(a[0]))

CK_MECHANISM ecdsaKeyPairGenMech = { CKM_EC_KEY_PAIR_GEN, NULL_PTR, 0 };

CK_OBJECT_HANDLE hPublicKey_Ecdsa = NULL_PTR;          //
CK_OBJECT_HANDLE hPrivateKey_Ecdsa = NULL_PTR;        //

/*****
*
*****/
printf("\nGenerating ECDSA key pair...\n");

rv = pFunctionList->C_GenerateKeyPair(hSession, &ecdsaKeyPairGenMech,
    publicKeyTemplate, arraysize(publicKeyTemplate),
    privateKeyTemplate, arraysize(privateKeyTemplate),
    &hPublicKey_Ecdsa, &hPrivateKey_Ecdsa);

if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
```

## Генерация секретного ключа

### Атрибуты ключевых объектов

Все поддерживаемые устройствами Рутокен атрибуты объектов представлены в разделе [Объекты секретных ключей](#).

### Поддерживаемые типы ключей

Устройства Рутокен поддерживают следующие типы секретных ключей (CK\_KEY\_TYPE) :

- CKK\_GENERIC\_SECRET для абстрактных ключей произвольной длины,
- CKK\_GOST28147 для ключей ГОСТ 28147-89,
- CKK\_MAGMA для использования в алгоритмах шифрования ГОСТ 34.12-2018 (ГОСТ Р 34.12-2015) с длиной блока 64 бит,
- CKK\_KUZNECHIK для использования в алгоритмах шифрования ГОСТ 34.12-2018 (ГОСТ Р 34.12-2015) с длиной блока 128 бит,
- CKK\_MAGMA\_TWIN\_KEY для использования в алгоритме экспорта и импорта ключей Р 1323565.1.017-2018, построенном на основании блочного шифра «Магма»,
- CKK\_KUZNECHIK\_TWIN\_KEY для использования в алгоритме экспорта и импорта ключей Р 1323565.1.017-2018, построенном на основании блочного шифра «Кузнечик»,

### Примеры шаблона секретного ключа

## Шаблон секретного ключа ГОСТ 28147-89

```
CK_OBJECT_CLASS ocSecKey          = CKO_SECRET_KEY;
CK_UTF8CHAR      SecKeyLabel[]    = {"GOST Secret Key"};
CK_BYTE          SecKeyID[]       = {"GOST Secret Key"};
CK_KEY_TYPE      KeyType          = CKK_GOST28147;
CK_BBOOL         bTrue            = CK_TRUE;
CK_BBOOL         bFalse           = CK_FALSE;

/* A 28147-89 */
CK_BYTE          GOST28147params[] = { 0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01 };

CK_ATTRIBUTE attrGOST28147_89SecKey[] =
{
    { CKA_CLASS, &ocSecKey, sizeof(ocSecKey)},           // 28147-89
    { CKA_LABEL, &SecKeyLabel, sizeof(SecKeyLabel) - 1}, //
    { CKA_ID, &SecKeyID, sizeof(SecKeyID) - 1},         //
    { CKA_KEY_TYPE, &KeyType, sizeof(KeyType)},         //
    { CKA_ENCRYPT, &bTrue, sizeof(bTrue)},               //
    { CKA_DECRYPT, &bTrue, sizeof(bTrue)},               //
    { CKA_TOKEN, &bTrue, sizeof(bTrue)},                 //
    { CKA_PRIVATE, &bFalse, sizeof(bFalse)},            //
    { CKA_GOST28147_PARAMS, GOST28147params, sizeof(GOST28147params)} //
};
```



## Шаблоны секретных ключей для ГОСТ Р 34.12-2015

```
CK_OBJECT_CLASS secretKeyObject = CKO_SECRET_KEY;
CK_UTF8CHAR secretKeyKuznechikLabel[] = { "Sample Kuznechik Secret Key (Aktiv Co.)" };
CK_UTF8CHAR secretKeyMagmaLabel[] = { "Sample Magma Secret Key (Aktiv Co.)" };
CK_BYTE secretKeyKuznechikId[] = { "Kuznechik Secret Key ID (Aktiv Co.)" };
CK_BYTE secretKeyMagmaId[] = { "Magma Secret Key ID (Aktiv Co.)" };
CK_KEY_TYPE keyTypeKuznechik = CKK_KUZNECHIK;
CK_KEY_TYPE keyTypeMagma = CKK_MAGMA;
CK_BBOOL attributeTrue = CK_TRUE;
CK_BBOOL attributeFalse = CK_FALSE;

/*****
 *
 *****/
CK_ATTRIBUTE kuznechikKeyTemplate[] =
{
    { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject) }, // -
    { CKA_LABEL, &secretKeyKuznechikLabel, sizeof(secretKeyKuznechikLabel) - 1 }, //
    { CKA_ID, &secretKeyKuznechikId, sizeof(secretKeyKuznechikId) - 1 }, //
    { CKA_KEY_TYPE, &keyTypeKuznechik, sizeof(keyTypeKuznechik) }, // -
    { CKA_ENCRYPT, &attributeTrue, sizeof(attributeTrue) }, //
    { CKA_DECRYPT, &attributeTrue, sizeof(attributeTrue) }, //
    { CKA_SIGN, &attributeFalse, sizeof(attributeFalse) }, // MAC
    { CKA_VERIFY, &attributeFalse, sizeof(attributeFalse) }, // MAC
    { CKA_TOKEN, &attributeTrue, sizeof(attributeTrue) }, //
    { CKA_PRIVATE, &attributeTrue, sizeof(attributeTrue) }, //
};

/*****
 *
 *****/
CK_ATTRIBUTE magmaKeyTemplate[] =
{
    { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject) }, // -
    { CKA_LABEL, &secretKeyMagmaLabel, sizeof(secretKeyMagmaLabel) - 1 }, //
    { CKA_ID, &secretKeyMagmaId, sizeof(secretKeyMagmaId) - 1 }, //
    { CKA_KEY_TYPE, &keyTypeMagma, sizeof(keyTypeMagma) }, // -
    { CKA_ENCRYPT, &attributeTrue, sizeof(attributeTrue) }, //
    { CKA_DECRYPT, &attributeTrue, sizeof(attributeTrue) }, //
    { CKA_SIGN, &attributeFalse, sizeof(attributeFalse) }, // MAC
    { CKA_VERIFY, &attributeFalse, sizeof(attributeFalse) }, // MAC
    { CKA_TOKEN, &attributeTrue, sizeof(attributeTrue) }, //
    { CKA_PRIVATE, &attributeTrue, sizeof(attributeTrue) }, //
};
```

## Поддерживаемые механизмы генерации ключей

Устройства Рутокен поддерживают следующие механизмы генерации секретного ключа:

- `СКМ_GOST28147_KEY_GEN` для генерации секретного ключа ГОСТ 28147-89 (библиотекой `тPKCS11ECP`),
- `СКМ_GOST_KEY_GEN` для генерации секретного ключа ГОСТ 28147-89 (библиотекой `тPKCS11`).
- `СКМ_KUZNECHIK_KEY_GEN` для генерации секретного ключа ГОСТ 34.12-2018 (ГОСТ Р 34.12-2015), используемого в алгоритмах шифрования с длиной блока 64 бит.
- `СКМ_MAGMA_KEY_GEN` для генерации секретного ключа ГОСТ 34.12-2018 (ГОСТ Р 34.12-2015), используемого в алгоритмах шифрования с длиной блока 128 бит.

## Пример генерации секретного ключа

Для генерации секретного ключа предназначена функция `C_GenerateKey()`, в которую передается механизм генерации и шаблон ключа.

Предварительно должна быть открыта сессия чтения/записи с авторизацией с правами Пользователя.

## Генерация симметричного ключа ГОСТ 28147-89

```
/* */
#define                arraysize(a)    (sizeof(a)/sizeof(a[0]))

CK_MECHANISM    KeyGenMech            = {CKM_GOST28147_KEY_GEN, NULL_PTR, 0}; //    28147-89.
CKM_KUZNECHIK_KEY_GEN    CKM_MAGMA_KEY_GEN

CK_OBJECT_HANDLE hSecKey              = NULL_PTR;                                //    c

...
printf("\n Generating key");
rv = pFunctionList->C_GenerateKey(hSession,                                     //
                                  &KeyGenMech,                               //
                                  attrGOST28147_89SecKey,                       //
                                  arraysize(attrGOST28147_89SecKey),           //
                                  &hSecKey);                                     //

if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
```

## Выработка сеансового симметричного ключа

Функция `C_DeriveKey()` позволяет получить одинаковый симметричный ключ для сеансовой связи на каждой из сторон, имея закрытый ключ одной стороны и открытый ключ другой стороны.

Устройства Рутокен поддерживают следующие механизмы согласования ключей:

- `CKM_GOSTR3410_DERIVE` для алгоритма VKO GOST R 34.10-2001;
- `CKM_GOSTR3410_12_DERIVE` для алгоритма VKO GOST R 34.10-2012 (256 и 512 бит).
- `CKM_KDF_TREE_GOSTR3411_2012_256` для алгоритма `KDF_TREE_GOSTR3411_2012_256`
- `CKM_VENDOR_GOST_KEG` для алгоритма Kег определенного в стандарте ГОСТ Р 1323565.1.020-2018

## VKO GOST R 34.10-2001

Выработанный общий ключ согласно VKO GOST R 34.10-2001 может быть возвращен в одном из следующих форматов:

- не диверсифицированный Key Encryption Key (KEK)
- KEK, диверсифицированный по RFC-4357, п.6.5.

Параметры механизма `CKM_GOSTR3410_DERIVE` задаются структурой `CK_GOSTR3410_DERIVE_PARAMS`, которая имеет следующие поля:

### структура `CK_GOSTR3410_DERIVE_PARAMS`

```
typedef struct CK_GOSTR3410_DERIVE_PARAMS{
    CK_EC_KDF_TYPE kdf;
    CK_BYTE_PTR pPublicData;
    CK_ULONG ulPublicDataLen;
    CK_BYTE_PTR pUKM;
    CK_ULONG ulUKMLen;
} CK_GOSTR3410_DERIVE_PARAMS;
```

1. `kdf` – идентификатор механизма диверсификации. Может быть использован один из двух механизмов:
  - `CKD_NULL` - нет диверсификации
  - `CKD_CPDIVERSIFY_KDF`
2. `pPublicData` – открытый ключ получателя
3. `pUKM` – имитовставка

## VKO GOST R 34.10-2012 (256 бит и 512 бит)

Выработанный общий ключ согласно VKO GOST R 34.10-2012 может быть возвращен только в формате не диверсифицированного Key Encryption Key (KEK).

Параметры механизма CKM\_GOSTR3410\_12\_DERIVE задаются байтовым массивом, который имеет следующую структуру:

1. 4 байта (little-endian, т.е. младшие байты сначала) представляют собой значение KDF. Значение определяет механизм диверсификации:
  - CKD\_NULL - нет диверсификации
  - CKM\_KDF\_4357
  - CKD\_CPDIVERSIFY\_KDF
  - CKM\_KDF\_GOSTR3411\_2012\_256
2. 4 байта (little-endian) задают длину открытого ключа в байтах. (для 256 бит – это 64. Для 512 бит – 128)
3. открытый ключ (n-байтовый вектор в little-endian), длина которого определена предыдущим полем
4. 4 байта (little-endian) задают длину UKM (8 байт)
5. UKM (n-байтовый вектор в little-endian), длина определена выше.

## KDF\_TREE\_GOSTR3411\_2012\_256

Механизм CKM\_KDF\_TREE\_GOSTR3411\_2012\_256 позволяет вырабатывать секретные и двойственные ключи из уже имеющегося секретного ключа. Этот механизм может применяться к ключу, выработанному с помощью алгоритма VKO.

Механизм может применяться к ключам шифрования СКК\_GOST28147, СКК\_KUZNYECHIK, СКК\_MAGMA и СКК\_GENERIC\_SECRET.

Результатом работы этого механизма могут быть ключи типа СКК\_GOST28147, СКК\_KUZNYECHIK, СКК\_MAGMA, СКК\_GENERIC\_SECRET, СКК\_KUZNYECHIK\_TWIN\_KEY и СКК\_MAGMA\_TWIN\_KEY.

Для задания параметров механизма используется структура CK\_KDF\_TREE\_GOST\_PARAMS :

### структура CK\_KDF\_TREE\_GOST\_PARAMS

```
typedef struct CK_KDF_TREE_GOST_PARAMS {
    CK_ULONG ulLabelLength;
    CK_BYTE_PTR pLabel;
    CK_ULONG ulSeedLength;
    CK_BYTE_PTR pSeed;
    CK_ULONG ulR;
    CK_ULONG ulL;
    CK_ULONG ulOffset;
} CK_KDF_TREE_GOST_PARAMS;
```

1. pLabel, pSeed – строки задающие параметры label и seed.
2. ulR – количество байт в счетчике итераций, с возможными значениями 1, 2, 3, 4.
3. ulL – необходимая байтовая длина вырабатываемого ключевого материала.
4. ulOffset – байтовое смещение, в последовательности ключевого материала, начиная с которого полученные байты используются для получения диверсифицированного ключа.

## CKM\_VENDOR\_GOST\_KEG

Механизм CKM\_VENDOR\_GOST\_KEG позволяет вырабатывать двойственный ключ из закрытого ключа отправителя и открытого ключа получателя.

Механизм может применяться к ключам ГОСТ Р 34.10-2012 с длиной ключа 256 и 512 бит.

Результатом работы этого механизма могут быть ключи типа СКК\_KUZNYECHIK\_TWIN\_KEY и СКК\_MAGMA\_TWIN\_KEY.

Для задания параметров механизма используется структура CK\_VENDOR\_GOST\_KEG\_PARAMS:

## структура CK\_VENDOR\_GOST\_KEG\_PARAMS

```
typedef struct CK_VENDOR_GOST_KEG_PARAMS {
    CK_BYTE_PTR pPublicData;
    CK_ULONG ulPublicDataLen;
    CK_BYTE_PTR pUKM;
    CK_ULONG ulUKMLen;
} CK_VENDOR_GOST_KEG_PARAMS;
```

1. pPublicData – открытый ключ получателя
2. pUKM – имитовставка.

## Пример выработки общего ключа парной связи по алгоритму VKO GOST R 34.10-2012

### Выработка общего ключа парной связи по схеме ключевого обмена VKO GOST 34.10-2001 на стороне отправителя

```
/* */
#define UKM_LENGTH 8

#define DERIVE_PARAMS_256_LENGTH 84

/*****
 * VKO GOST R 34.10-2012-256*
 * , UKM *
 *****/
CK_BYTE deriveParameters2012_256[DERIVE_PARAMS_256_LENGTH] = { 0x00, };

const CK_ULONG keyLengthOffset = 4; //
const CK_ULONG publicKeyValueOffset = 8; //
const CK_ULONG ukmLengthOffset = 72; // UKM
const CK_ULONG ukmDataOffset = 76; // UKM

/*****
 *
 *****/
void ulongToBuffer(CK_BYTE_PTR buffer, CK_ULONG value)
{
    buffer[0] = value & 0xFF;
    buffer[1] = (value >> 8) & 0xFF;
    buffer[2] = (value >> 16) & 0xFF;
    buffer[3] = (value >> 24) & 0xFF;
}

/* VKO GOST R 34.10-2012 */
CK_MECHANISM gostR3410_12DerivationMech = { CKM_GOSTR3410_12_DERIVE, NULL_PTR, 0 };

/* */
CK_BYTE cbPubRecipientKey[] = { 0xFF, 0x8D, 0xAB, 0x7F, 0x1C, 0x0B, 0x74, 0xA5, 0xAD, 0x7F, 0x0B, 0x5F, 0x8D,
0x5B, 0x3C, 0x44,
0x58, 0x37, 0x98, 0xC9, 0x25, 0x86, 0x40, 0x7E,
0xEC, 0x6E, 0xAF, 0x00, 0xCB, 0x44, 0x65, 0xA5,
0x22, 0x9A, 0x53, 0x56, 0x32, 0x97, 0x35, 0x80,
0x99, 0xCA, 0x1E, 0x17, 0x21, 0x3A, 0x96, 0x0E,
0x21, 0xFB, 0xC6, 0x0F, 0x25, 0x5B, 0x5D, 0x99,
0x4E, 0xC4, 0x5C, 0x42, 0x08, 0x7D, 0x06, 0x04 };

CK_OBJECT_CLASS ocSecKey = CKO_SECRET_KEY;
CK_UTF8CHAR DerivedKeyLabel[] = { "Derived Key" };
CK_BYTE SecKeyID[] = { "GOST Secret Key" };
CK_KEY_TYPE KeyType = CKK_GOST28147;
CK_BBOOL bTrue = CK_TRUE;
CK_BBOOL bFalse = CK_FALSE;

/* */
```

```

CK_ATTRIBUTE attrGOST28147DerivedKey[] =
{
    { CKA_CLASS, &ocSecKey, sizeof(ocSecKey)},           //      28147-89
    { CKA_LABEL, &DerivedKeyLabel, sizeof(DerivedKeyLabel) - 1}, //
    { CKA_KEY_TYPE, &KeyType, sizeof(KeyType)},           //
    { CKA_TOKEN, &bFalse, sizeof(bFalse)},               //
    { CKA_MODIFIABLE, &bTrue, sizeof(bTrue)},             //
    { CKA_PRIVATE, &bFalse, sizeof(bFalse)},             //
    { CKA_EXTRACTABLE, &bTrue, sizeof(bTrue)},           //
    { CKA_SENSITIVE, &bFalse, sizeof(bFalse)}            //
};

CK_ATTRIBUTE          attrDerivedKeyValue = { CKA_VALUE, NULL_PTR, 0 };           //      CK_ATTRIBUTE
CK_VALUE
CK_BYTE              ukm
[UKM_LENGTH];
CK_OBJECT_HANDLE     hDerivedKey = NULL_PTR;                                     //
CK_OBJECT_HANDLE     hObject;                                                  //

...

/*****
*      CK_GOSTR3410_DERIVE_PARAMS      *
*
*****/
rv = pFunctionList->C_GenerateRandom(hSession, ukm, sizeof(ukm));
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto exit;
}

/*****
*      CK_MECHANISM ,      *
*
*****/
ulongToBuffer(deriveParameters2012_256, CKM_KDF_GOSTR3411_2012_256);
ulongToBuffer(deriveParameters2012_256 + keyLengthOffset, sizeof(cbPubRecipientKey));
memcpy(deriveParameters2012_256 + publicKeyValueOffset, cbPubRecipientKey, sizeof(cbPubRecipientKey));
ulongToBuffer(deriveParameters2012_256 + ukmLengthOffset, sizeof(ukm));
memcpy(deriveParameters2012_256 + ukmDataOffset, ukm, sizeof(ukm));
gostR3410_12DerivationMech.pParameter = deriveParameters2012_256;
gostR3410_12DerivationMech.ulParameterLen = sizeof(deriveParameters2012_256);

/*      28147-89      */
printf("C_DeriveKey");
rv = pFunctionList->C_DeriveKey(hSession,           //
    &gostR3410_12DerivationMech,                   //
    hPrivateKey,                                   //
    attrGOST28147DerivedKey,                       //
    arraysize(attrGOST28147DerivedKey),           //
    &hDerivedKey);                                 //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto exit;
}
printf(" -> OK\n");

/*      CKA_VALUE*/
printf("Getting object value size");
rv = pFunctionList->C_GetAttributeValue(hSession, //
    hDerivedKey,                                   //
    &attrDerivedKeyValue,                         //
    1);                                             //

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
}

```

```

        goto exit;
    }
    printf(" -> OK\n");

    /*      */
    attrDerivedKeyValue.pValue = (CK_BYTE*)malloc(attrDerivedKeyValue.ulValueLen);
    if (attrDerivedKeyValue.pValue == NULL)
    {
        printf("Memory allocation for attrDerivedKeyValue failed! \n");
        goto exit;
    }
    memset(attrDerivedKeyValue.pValue,
           0,
           (attrDerivedKeyValue.ulValueLen * sizeof(CK_BYTE)));

    /*      28147-89 */
    printf("Getting object value");
    rv = pFunctionList->C_GetAttributeValue(hSession,          //
        hDerivedKey,
        &attrDerivedKeyValue,                               //
        1);                                                  //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        goto exit;
    }
    printf(" -> OK\n");
    /*      28147-89 */
    printf("Derived key data is:\n");
    for (size_t i = 0; i < attrDerivedKeyValue.ulValueLen; i++)
    {
        printf("%02X ", ((CK_BYTE_PTR) attrDerivedKeyValue.pValue)[i]);
        if ((i + 1) % 8 == 0)
            printf("\n");
    }

    exit:
    if (attrDerivedKeyValue.pValue)
    {
        free(attrDerivedKeyValue.pValue);
        attrDerivedKeyValue.pValue = NULL_PTR;
        attrDerivedKeyValue.ulValueLen = 0;
    }

    if (rv != CKR_OK)
    {
        pFunctionList->C_DestroyObject(hSession,
            hDerivedKey);
        hDerivedKey = NULL_PTR;
    }
    if (rv != CKR_OK)
        printf("\nDeriving failed!\n\n");
    else
        printf("Deriving has been completed successfully.\n\n");

```

## Пример выработки двойственного ключа по алгоритму Кег

### Выработка двойственного ключа по алгоритму Кег

```

/*      */
#define UKM_KEG_LENGTH                24

CK_UTF8CHAR derivedKuznechikTwinKeyLabel[] = { "Derived Kuznechik twin key" };
CK_UTF8CHAR derivedMagmaTwinKeyLabel[] = { "Derived Magma twin key" };
CK_OBJECT_CLASS secretKeyObject = CKO_SECRET_KEY;
CK_KEY_TYPE keyTypeKuznechikTwin = CKK_KUZNECHIK_TWIN_KEY;
CK_KEY_TYPE keyTypeMagmaTwin = CKK_MAGMA_TWIN_KEY;

```

```

CK_BBOOL attributeTrue = CK_TRUE;
CK_BBOOL attributeFalse = CK_FALSE;

/*      */
CK_BYTE cbPubRecipientKey[] = { 0xFF, 0x8D, 0xAB, 0x7F, 0x1C, 0x0B, 0x74, 0xA5, 0xAD, 0x7F, 0x0B, 0x5F, 0x8D,
0x5B, 0x3C, 0x44,
0x58, 0x37, 0x98, 0xC9, 0x25, 0x86, 0x40, 0x7E,
0xEC, 0x6E, 0xAF, 0x00, 0xCB, 0x44, 0x65, 0xA5,
0x22, 0x9A, 0x53, 0x56, 0x32, 0x97, 0x35, 0x80,
0x99, 0xCA, 0x1E, 0x17, 0x21, 0x3A, 0x96, 0x0E,
0x21, 0xFB, 0xC6, 0x0F, 0x25, 0x5B, 0x5D, 0x99,
0x4E, 0xC4, 0x5C, 0x42, 0x08, 0x7D, 0x06, 0x04 };

/*****
*
*****/
CK_ATTRIBUTE derivedTwinKeyTemplate[] =
{
    { CKA_LABEL, &derivedKuznechikTwinKeyLabel, sizeof(derivedKuznechikTwinKeyLabel) - 1}, //
    { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject)
}, // -
    { CKA_KEY_TYPE, &keyTypeKuznechikTwin, sizeof
(keyTypeKuznechikTwin)}, // -
    { CKA_TOKEN, &attributeFalse, sizeof
(attributeFalse)}, //
    { CKA_MODIFIABLE, &attributeTrue, sizeof
(attributeTrue)}, //
    { CKA_PRIVATE, &attributeTrue, sizeof
(attributeTrue)}, //
    { CKA_EXTRACTABLE, &attributeTrue, sizeof
(attributeTrue)}, //
    { CKA_SENSITIVE, &attributeFalse, sizeof
(attributeFalse)} //
};

/*****
*
*****/
//CK_ATTRIBUTE derivedTwinKeyTemplate[] =
//{
//    { CKA_LABEL, &derivedMagmaTwinKeyLabel, sizeof(derivedMagmaTwinKeyLabel) - 1}, //
//    { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject)
//}, // -
//    { CKA_KEY_TYPE, &keyTypeMagmaTwin, sizeof
// (keyTypeMagmaTwin)}, // -
//    { CKA_TOKEN, &attributeFalse, sizeof
// (attributeFalse)}, //
//    { CKA_MODIFIABLE, &attributeTrue, sizeof
// (attributeTrue)}, //
//    { CKA_PRIVATE, &attributeTrue, sizeof
// (attributeTrue)}, //
//    { CKA_EXTRACTABLE, &attributeTrue, sizeof
// (attributeTrue)}, //
//    { CKA_SENSITIVE, &attributeFalse, sizeof
// (attributeFalse)} //
//};

CK_MECHANISM gostKegDerivationMech = { CKM_VENDOR_GOST_KEG, NULL_PTR, 0 };
CK_VENDOR_GOST_KEG_PARAMS keg256DeriveParams;
CK_ATTRIBUTE attrDerivedKeyValue = { CKA_VALUE, NULL_PTR, 0 }; // CK_ATTRIBUTE
CK_VALUE
CK_BYTE          ukm
[UKM_KEG_LENGTH]; // , UKM
CK_OBJECT_HANDLE hDerivedKey = NULL_PTR; //
CK_OBJECT_HANDLE
hObject; //

...

/*****

```

```

*      CK_VENDOR_GOST_KEG_PARAMS      *
*
*****/
rv = pFunctionList->C_GenerateRandom(hSession, ukm, sizeof(ukm));
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto exit;
}

/*****
*      CK_MECHANISM ,      *
*
*****/

keg256DeriveParams.pPublicData = cbPubRecipientKey;
keg256DeriveParams.ulPublicDataLen = sizeof(cbPubRecipientKey);
keg256DeriveParams.pUKM = ukm;
keg256DeriveParams.ulUKMLen = sizeof(ukm);

gostR3410_12DerivationMech.pParameter = keg256DeriveParams;
gostR3410_12DerivationMech.ulParameterLen = sizeof(keg256DeriveParams);

/*      28147-89      */
printf("C_DeriveKey");
rv = pFunctionList->C_DeriveKey(hSession,          //
    &gostR3410_12DerivationMech,          //
    hPrivateKey,          //
    derivedTwinKeyTemplate,          //
    arraysize(derivedTwinKeyTemplate),          //
    &hDerivedKey);          //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto exit;
}
printf(" -> OK\n");

/*      CKA_VALUE*/
printf("Getting object value size");
rv = pFunctionList->C_GetAttributeValue(hSession,          //
    hDerivedKey,          //
    &attrDerivedKeyValue,          //
    1);          //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto exit;
}
printf(" -> OK\n");

/*      */
attrDerivedKeyValue.pValue = (CK_BYTE*)malloc(attrDerivedKeyValue.ulValueLen);
if (attrDerivedKeyValue.pValue == NULL)
{
    printf("Memory allocation for attrDerivedKeyValue failed! \n");
    goto exit;
}
memset(attrDerivedKeyValue.pValue,
    0,
    (attrDerivedKeyValue.ulValueLen * sizeof(CK_BYTE)));

/*      28147-89 */
printf("Getting object value");
rv = pFunctionList->C_GetAttributeValue(hSession,          //
    hDerivedKey,          //
    &attrDerivedKeyValue,          //
    1);          //

```



```

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto exit;
}
printf(" -> OK\n");
/*      28147-89 */
printf("Derived key data is:\n");
for (size_t i = 0; i < attrDerivedKeyValue.ulValueLen; i++)
{
    printf("%02X ", ((CK_BYTE_PTR)attrDerivedKeyValue.pValue)[i]);
    if ((i + 1) % 8 == 0)
        printf("\n");
}

exit:
if (attrDerivedKeyValue.pValue)
{
    free(attrDerivedKeyValue.pValue);
    attrDerivedKeyValue.pValue = NULL_PTR;
    attrDerivedKeyValue.ulValueLen = 0;
}

if (rv != CKR_OK)
{
    pFunctionList->C_DestroyObject(hSession,
        hDerivedKey);
    hDerivedKey = NULL_PTR;
}
if (rv != CKR_OK)
printf("\nDeriving failed!\n\n");
else
printf("Deriving has been completed successfully.\n\n");

```

## Маскирование секретного ключа

Для получения сеансового ключ (CEK), зашифрованного на КЕК с имитовставкой CEK, необходимо после получения ключа функцией C\_DeriveKey() использовать функцию маскирования C\_WrapKey().

Для выработки сеансового ключ (CEK) в оперативной памяти токена, зашифрованного на КЕК с имитовставкой CEK, необходимо использовать функцию расширения C\_EX\_WrapKey().

Для маскирования (шифрования) симметричного ключа используются функция C\_WrapKey() для маскирования и C\_UnwrapKey() для обратной процедуры.

В этом примере мы генерируем случайным образом сессионный ключ (CEK), а затем маскируем его общим ключом КЕК, полученным из функции C\_Derive().

### Маскирование общего выработанного ключа

```

/*      */
#define UKM_LENGTH                8
/*      28147-89 */
#define GOST_28147_KEY_SIZE      0x20

/*      A      28147-89 */
CK_BYTE GOST28147params[] = { 0x06, 0x07, 0x2a, 0x85, 0x03, 0x02, 0x02, 0x1f, 0x01 };

CK_BYTE ukm[UKM_LENGTH] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08 }; //

/*      /      */
CK_MECHANISM    ckmWrapMech = { CKM_GOST28147_KEY_WRAP, NULL_PTR, 0 };

CK_OBJECT_CLASS ocSecKey = CKO_SECRET_KEY;
CK_UTF8CHAR     WrapKeyLabel[] = { "GOST Wrapped Key" };
CK_UTF8CHAR     UnWrapKeyLabel[] = { "GOST Unwrapped Key" };
CK_KEY_TYPE     KeyType = CKK_GOST28147;
CK_BBOOL        bTrue = CK_TRUE;

```

```

CK_BOOL                bFalse = CK_FALSE;

/* */
CK_ATTRIBUTE attrGOST28147KeyToWrap[] =
{
    { CKA_CLASS, &ocSecKey, sizeof(ocSecKey)}, // 28147-89
    { CKA_LABEL, &WrapKeyLabel, sizeof(WrapKeyLabel) - 1 }, //
    { CKA_KEY_TYPE, &KeyType, sizeof(KeyType)}, //
    { CKA_TOKEN, &bFalse, sizeof(bFalse)}, //
    { CKA_MODIFIABLE, &bTrue, sizeof(bTrue)}, //
    { CKA_PRIVATE, &bFalse, sizeof(bFalse)}, //
    { CKA_VALUE, NULL_PTR, 0}, //
    { CKA_EXTRACTABLE, &bTrue, sizeof(bTrue)}, //
    { CKA_SENSITIVE, &bFalse, sizeof(bFalse)} //
};

/* */
CK_ATTRIBUTE attrGOST28147UnwrappedKey[] =
{
    { CKA_CLASS, &ocSecKey, sizeof(ocSecKey)}, // 28147-89
    { CKA_LABEL, &UnWrapKeyLabel, sizeof(UnWrapKeyLabel) - 1}, //
    { CKA_KEY_TYPE, &KeyType, sizeof(KeyType)}, //
    { CKA_TOKEN, &bFalse, sizeof(bFalse)}, //
    { CKA_MODIFIABLE, &bTrue, sizeof(bTrue)}, //
    { CKA_PRIVATE, &bFalse, sizeof(bFalse)}, //
    { CKA_EXTRACTABLE, &bTrue, sizeof(bTrue)}, //
    { CKA_SENSITIVE, &bFalse, sizeof(bFalse)} //
};

/* CK_ATTRIBUTE CKA_VALUE */
CK_ATTRIBUTE attrValue = { CKA_VALUE, NULL_PTR, 0 };

CK_OBJECT_HANDLE hDerivedKey; //
CK_BYTE_PTR pbtSessionKey = NULL_PTR; // ,
CK_BYTE_PTR pbtWrappedKey = NULL_PTR; // ,
CK_ULONG ulWrappedKeySize = 0; // ,
CK_BYTE_PTR pbtUnwrappedKey = NULL_PTR; // ,
CK_ULONG ulUnwrappedKeySize = 0; // ,
CK_OBJECT_HANDLE hTempKey = NULL_PTR; // , /

...

/* */
ckmWrapMech.ulParameterLen = sizeof(ukm);
ckmWrapMech.pParameter = ukm;

/* */
GenerateRandomData(GOST_28147_KEY_SIZE, &pbtSessionKey);
for (int i = 0; i < arraysize(attrGOST28147KeyToWrap); i++)
    if (attrGOST28147KeyToWrap[i].type == CKA_VALUE)
    {
        attrGOST28147KeyToWrap[i].pValue = pbtSessionKey;
        attrGOST28147KeyToWrap[i].ulValueLen = GOST_28147_KEY_SIZE;
        break;
    }

/*****
*
*****/

/* , */
printf("Creating the GOST 28147-89 key to wrap");
rv = pFunctionList->C_CreateObject(hSession, // ,
    attrGOST28147KeyToWrap, //
    arraysize(attrGOST28147KeyToWrap), //
    &hTempKey); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto wrap_exit;
}

```

```

}
printf(" -> OK\n");

/* , */
printf("Defining wrapping key size");
rv = pFunctionList->C_WrapKey(hSession, // ,
    &ckmWrapMech, //
    hDerivedKey, // ,
    hTempKey, // ,
    NULL_PTR, //
    &ulWrappedKeySize); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto wrap_exit;
}
printf(" -> OK\n");

pbtWrappedKey = (CK_BYTE*)malloc(ulWrappedKeySize);
if (pbtWrappedKey == NULL)
{
    printf("Memory allocation for pbtWrappedKey failed! \n");
    goto wrap_exit;
}
memset(pbtWrappedKey,
    0,
    ulWrappedKeySize * sizeof(CK_BYTE));

/* */
printf("Wrapping key");
rv = pFunctionList->C_WrapKey(hSession, // ,
    &ckmWrapMech, //
    hDerivedKey, // ,
    hTempKey, // ,
    pbtWrappedKey, //
    &ulWrappedKeySize); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto wrap_exit;
}
printf(" -> OK\n");

/* , */
printf("Wrapped key data is:\n");
for (int i = 0; i < ulWrappedKeySize; i++)
{
    printf("%02X ", pbtWrappedKey[i]);
    if ((i + 1) % 9 == 0)
        printf("\n");
}

wrap_exit:
if (hTempKey)
{
    pFunctionList->C_DestroyObject(hSession,
        hTempKey);
    hTempKey = NULL_PTR;
}

if (rv == CKR_OK)
    printf("\nWrapping has been completed successfully.\n");
else
{
    printf("\nWrapping failed!\n");
    goto exit;
}

/*****

```

```

*
*
*****/
printf("Unwrapping key");
rv = pFunctionList->C_UnwrapKey(hSession, // ,

    &ckmWrapMech, //
    hDerivedKey, // ,
    pbtWrappedKey, //
    ulWrappedKeySize, //
    attrGOST28147UnwrappedKey, //
    arraysize(attrGOST28147UnwrappedKey), //
    &hTempKey); //

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto unwrap_exit;
}
printf(" -> OK\n");

/* */
printf("Getting unwrapped key value...\n");

/* CKA_VALUE */
printf("Getting object value size");
rv = pFunctionList->C_GetAttributeValue(hSession, // ,
    hTempKey, // ,
    &attrValue, // ,
    1); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto unwrap_exit;
}
printf(" -> OK\n");

/* */
attrValue.pValue = (CK_BYTE*)malloc(attrValue.ulValueLen);
if (attrValue.pValue == NULL)
{
    printf("Memory allocation for attrValue failed! \n");
    goto unwrap_exit;
}
memset(attrValue.pValue,
    0,
    (attrValue.ulValueLen * sizeof(CK_BYTE)));

/* CKA_VALUE */
printf("Getting object value");
rv = pFunctionList->C_GetAttributeValue(hSession, // ,
    hTempKey, // ,
    &attrValue, // ,
    1); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto unwrap_exit;
}
printf(" -> OK\n");
/* */
printf("Unwrapped key data:\n");
for (int i = 0; i < attrValue.ulValueLen; i++)
{
    printf("%02X ", ((CK_BYTE_PTR)attrValue.pValue)[i]);
    if ((i + 1) % 8 == 0)
        printf("\n");
}

unwrap_exit:
if (hTempKey)
{
    pFunctionList->C_DestroyObject(hSession,

```

```

        hTempKey);
        hTempKey = NULL_PTR;
    }
    if (rv == CKR_OK)
        printf("Unwrapping has been completed successfully.\n\n");
    else
    {
        printf("\nUnwrapping failed!\n\n");
        goto exit;
    }

    /*
     */
    if ((ulUnwrappedKeySize != GOST_28147_KEY_SIZE)
        || (memcmp(pbtSessionKey,
                  attrValue.pValue,
                  GOST_28147_KEY_SIZE) != 0))
        printf("\nThe unwrapped key is not equal to the session key!\n");
    else
        printf("The unwrapped key is equal to the session key.\n");

    exit:
    printf("Finish");

```

## Пример экспорта и импорта ключа по алгоритму Кехр15

### Маскирование общего выработанного ключа

```

/*
   KExp15
 */
CK_BYTE gostKuznechikKExp15Ukm[] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
/*
   KExp15
 */
CK_BYTE gostMagmaKExp15Ukm[] = { 0x00, 0x00, 0x00, 0x00 };

/*
   KExp15
 */
CK_MECHANISM gostKuznechikKExp15Mech = { CKM_KUZNECHIK_KEYP_15_WRAP, &gostKuznechikKExp15Ukm, sizeof
(gostKuznechikKExp15Ukm) };
/*
   KExp15
 */
CK_MECHANISM gostMagmaKExp15Mech = { CKM_MAGMA_KEYP_15_WRAP, &gostMagmaKExp15Ukm, sizeof(gostMagmaKExp15Ukm) };

CK_MECHANISM_PTR gostKExp15Mech = &gostKuznechikKExp15Mech;
//CK_MECHANISM gostKExp15Mech = gostMagmaKExp15Mech;

/* DEMO-
   */
CK_UTF8CHAR sessionKeyLabel[] = { "GOST 28147-89 key to wrap" };
CK_UTF8CHAR sessionKuznechikKeyLabel[] = { "Kuznechik key to wrap" };
CK_UTF8CHAR sessionMagmaKeyLabel[] = { "Magma key to wrap" };
CK_OBJECT_CLASS secretKeyObject = CKO_SECRET_KEY;
CK_KEY_TYPE keyTypeGost28147 = CKK_GOST28147;
CK_KEY_TYPE keyTypeKuznechik = CKK_KUZNECHIK;
CK_KEY_TYPE keyTypeMagma = CKK_MAGMA;
CK_BBOOL attributeTrue = CK_TRUE;
CK_BBOOL attributeFalse = CK_FALSE;

/*****
 *
 *****/
CK_ATTRIBUTE sessionKeyTemplate[] =
{
    { CKA_LABEL, &sessionKuznechikKeyLabel, sizeof(sessionKuznechikKeyLabel) - 1 }, //
    { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject)
}, // -
    { CKA_KEY_TYPE, &keyTypeKuznechik, sizeof(keyTypeKuznechik)}, //
    -
    { CKA_TOKEN, &attributeFalse, sizeof
(attributeFalse)}, //
    { CKA_MODIFIABLE, &attributeTrue, sizeof
(attributeTrue)}, //
    { CKA_PRIVATE, &attributeTrue, sizeof

```

```

(attributeTrue)}, //
    { CKA_VALUE, NULL_PTR,
0},
//
    { CKA_EXTRACTABLE, &attributeTrue, sizeof
(attributeTrue)}, //
    { CKA_SENSITIVE, &attributeFalse, sizeof
(attributeFalse)} //
};

/*****
*
*****/
//CK_ATTRIBUTE sessionKeyTemplate[] =
//{
//    { CKA_LABEL, &sessionMagmaKeyLabel, sizeof(sessionMagmaKeyLabel) - 1 },//
//    { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject) }, // -
//    { CKA_KEY_TYPE, &keyTypeMagma, sizeof(keyTypeMagma)}, // -
//    { CKA_TOKEN, &attributeFalse, sizeof(attributeFalse)}, //
//    { CKA_MODIFIABLE, &attributeTrue, sizeof(attributeTrue)}, //
//    { CKA_PRIVATE, &attributeTrue, sizeof(attributeTrue)}, //
//    { CKA_VALUE, NULL_PTR, 0}, //
//    { CKA_EXTRACTABLE, &attributeTrue, sizeof(attributeTrue)}, //
//    { CKA_SENSITIVE, &attributeFalse, sizeof(attributeFalse)} //
//};

/*****
*      28147-89      *
*****/
//CK_ATTRIBUTE sessionKeyTemplate[] =
//{
//    { CKA_LABEL, &sessionKeyLabel, sizeof(sessionKeyLabel) - 1 }, //
//    { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject) }, // -
//    { CKA_KEY_TYPE, &keyTypeGost28147, sizeof(keyTypeGost28147)}, // - 28147-89
//    { CKA_TOKEN, &attributeFalse, sizeof(attributeFalse)}, //
//    { CKA_MODIFIABLE, &attributeTrue, sizeof(attributeTrue)}, //
//    { CKA_PRIVATE, &attributeTrue, sizeof(attributeTrue)}, //
//    { CKA_VALUE, NULL_PTR, 0}, //
//    { CKA_EXTRACTABLE, &attributeTrue, sizeof(attributeTrue)}, //
//    { CKA_SENSITIVE, &attributeFalse, sizeof(attributeFalse)} //
//};

/*****
*
*****/
CK_ATTRIBUTE unwrapSessionKeyTemplate[] =
{
    { CKA_LABEL, &sessionKuznechikKeyLabel, sizeof(sessionKuznechikKeyLabel) - 1 }, //
    { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject)
}, // -
    { CKA_KEY_TYPE, &keyTypeKuznechik, sizeof(keyTypeKuznechik)}, //
-
    { CKA_TOKEN, &attributeFalse, sizeof
(attributeFalse)}, //
    { CKA_MODIFIABLE, &attributeTrue, sizeof
(attributeTrue)}, //
    { CKA_PRIVATE, &attributeTrue, sizeof
(attributeTrue)}, //
    { CKA_EXTRACTABLE, &attributeTrue, sizeof
(attributeTrue)}, //
    { CKA_SENSITIVE, &attributeFalse, sizeof
(attributeFalse)} //
};

/*****
*
*****/
//CK_ATTRIBUTE sessionKeyTemplate[] =
//{
//    { CKA_LABEL, &sessionMagmaKeyLabel, sizeof(sessionMagmaKeyLabel) - 1 },//

```

```

//      { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject) },           // -
//      { CKA_KEY_TYPE, &keyTypeMagma, sizeof(keyTypeMagma)},               // -
//      { CKA_TOKEN, &attributeFalse, sizeof(attributeFalse)},             //
//      { CKA_MODIFIABLE, &attributeTrue, sizeof(attributeTrue)},          //
//      { CKA_PRIVATE, &attributeTrue, sizeof(attributeTrue)},            //
//      { CKA_EXTRACTABLE, &attributeTrue, sizeof(attributeTrue)},         //
//      { CKA_SENSITIVE, &attributeFalse, sizeof(attributeFalse)}          //
//};

/*****
*      28147-89      *
*****/
//CK_ATTRIBUTE sessionKeyTemplate[] =
//{
//      { CKA_LABEL, &sessionKeyLabel, sizeof(sessionKeyLabel) - 1 },       //
//      { CKA_CLASS, &secretKeyObject, sizeof(secretKeyObject) },          // -
//      { CKA_KEY_TYPE, &keyTypeGost28147, sizeof(keyTypeGost28147)},      // - 28147-89
//      { CKA_TOKEN, &attributeFalse, sizeof(attributeFalse)},            //
//      { CKA_MODIFIABLE, &attributeTrue, sizeof(attributeTrue)},          //
//      { CKA_PRIVATE, &attributeTrue, sizeof(attributeTrue)},            //
//      { CKA_EXTRACTABLE, &attributeTrue, sizeof(attributeTrue)},         //
//      { CKA_SENSITIVE, &attributeFalse, sizeof(attributeFalse)}          //
//};

/*      CK_ATTRIBUTE      CKA_VALUE */
CK_ATTRIBUTE attrValue = { CKA_VALUE, NULL_PTR, 0 };

CK_OBJECT_HANDLE hDerivedTwinKey;
CK_BYTE_PTR pbtSessionKey = NULL_PTR;
CK_BYTE_PTR pbtWrappedKey = NULL_PTR;
CK_ULONG ulWrappedKeySize = 0;
CK_BYTE_PTR pbtUnwrappedKey = NULL_PTR;
CK_ULONG ulUnwrappedKeySize = 0;
CK_OBJECT_HANDLE hTempKey = NULL_PTR;

...

/*      */
GenerateRandomData(GOST_28147_KEY_SIZE, &pbtSessionKey);
for (int i = 0; i < arraysize(sessionKeyTemplate); i++)
    if (sessionKeyTemplate[i].type == CKA_VALUE)
    {
        sessionKeyTemplate[i].pValue = pbtSessionKey;
        sessionKeyTemplate[i].ulValueLen = GOST_KUZNECHIK_KEY_SIZE;
        break;
    }

/*****
*      *
*****/

/*      */
printf("Creating the GOST 28147-89 key to wrap");
rv = pFunctionList->C_CreateObject(hSession,
    sessionKeyTemplate,
    arraysize(sessionKeyTemplate),
    &hTempKey);
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto wrap_exit;
}
printf(" -> OK\n");

/*      */
printf("Defining wrapping key size");
rv = pFunctionList->C_WrapKey(hSession,
    gostKExpl5Mech,
    hDerivedTwinKey,
    hTempKey,
    NULL_PTR,

```

```

        &ulWrappedKeySize); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto wrap_exit;
}
printf(" -> OK\n");

pbtWrappedKey = (CK_BYTE*)malloc(ulWrappedKeySize);
if (pbtWrappedKey == NULL_PTR)
{
    printf("Memory allocation for pbtWrappedKey failed! \n");
    goto wrap_exit;
}
memset(pbtWrappedKey,
        0,
        ulWrappedKeySize * sizeof(CK_BYTE));

/*      */
printf("Wrapping key");
rv = pFunctionList->C_WrapKey(hSession, // ,
    gostKExp15Mech, //
    hDerivedTwinKey, // ,
    hTempKey, // // ,
    pbtWrappedKey, //
    &ulWrappedKeySize); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto wrap_exit;
}
printf(" -> OK\n");

/*      */
printf("Wrapped key data is:\n");
for (int i = 0; i < ulWrappedKeySize; i++)
{
    printf("%02X ", pbtWrappedKey[i]);
    if ((i + 1) % 9 == 0)
        printf("\n");
}

wrap_exit:
if (hTempKey)
{
    pFunctionList->C_DestroyObject(hSession,
        hTempKey);
    hTempKey = NULL_PTR;
}

if (rv == CKR_OK)
printf("\nWrapping has been completed successfully.\n");
else
{
    printf("\nWrapping failed!\n");
    goto exit;
}

/*****
*
*****/
printf("Unwrapping key");
rv = pFunctionList->C_UnwrapKey(hSession, // ,
    gostKExp15Mech, //
    hDerivedTwinKey, // ,
    pbtWrappedKey, // //
    ulWrappedKeySize, //
    unwrapSessionKeyTemplate, //

```



```

        arraysize(unwrapSessionKeyTemplate), //
        &hTempKey); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto unwrap_exit;
}
printf(" -> OK\n");

/*      */
printf("Getting unwrapped key value...\n");

/*      CKA_VALUE */
printf("Getting object value size");
rv = pFunctionList->C_GetAttributeValue(hSession, // ,
    hTempKey, // ,
    &attrValue, // ,
    1); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto unwrap_exit;
}
printf(" -> OK\n");

/*      */
attrValue.pValue = (CK_BYTE*)malloc(attrValue.ulValueLen);
if (attrValue.pValue == NULL_PTR)
{
    printf("Memory allocation for attrValue failed! \n");
    goto unwrap_exit;
}
memset(attrValue.pValue,
    0,
    (attrValue.ulValueLen * sizeof(CK_BYTE)));

/*      CKA_VALUE */
printf("Getting object value");
rv = pFunctionList->C_GetAttributeValue(hSession, // ,
    hTempKey, // ,
    &attrValue, // ,
    1); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    goto unwrap_exit;
}
printf(" -> OK\n");
/*      */
printf("Unwrapped key data:\n");
for (int i = 0; i < attrValue.ulValueLen; i++)
{
    printf("%02X ", ((CK_BYTE_PTR)attrValue.pValue)[i]);
    if ((i + 1) % 8 == 0)
        printf("\n");
}

unwrap_exit:
if (hTempKey)
{
    pFunctionList->C_DestroyObject(hSession,
        hTempKey);
    hTempKey = NULL_PTR;
}
if (rv == CKR_OK)
printf("Unwrapping has been completed successfully.\n\n");
else
{
    printf("\nUnwrapping failed!\n\n");
    goto exit;
}

```

```

/*      */
if ((ulUnwrappedKeySize != GOST_KUZNECHIK_KEY_SIZE)
    || (memcmp(pbtSessionKey,
              attrValue.pValue,
              GOST_KUZNECHIK_KEY_SIZE) != 0))
    printf("\nThe unwrapped key is not equal to the session key!\n");
else
    printf("The unwrapped key is equal to the session key.\n");

exit:
printf("Finish");

```

## Вычисление значения хеш-функции

### Поддерживаемые механизмы

Устройства Рутокен поддерживают следующие механизмы хеширования:

- СКМ\_MD2 для хеширования алгоритмом MD2 (только программно),
- СКМ\_MD5 для хеширования алгоритмом MD5 (только программно),
- СКМ\_SHA\_1 для хеширования алгоритмом SHA-1 (только программно),
- СКМ\_GOSTR3411 для хеширования алгоритмом ГОСТ Р 34.11.94 (программно и аппаратно),
- СКМ\_GOSTR3411\_12\_256 для хеширования алгоритмом ГОСТ Р 34.11.2012 с длиной значения 256 бит (только аппаратно),
- СКМ\_GOSTR3411\_12\_512 для хеширования алгоритмом ГОСТ Р 34.11.2012 с длиной закрытого ключа 512 бит (только аппаратно).

### Хеширование данных

Для хеширования данных служат функции `C_DigestInit()` и `C_Digest()`. Сначала операцию хеширования нужно инициализировать через `C_DigestInit()`, передав в нее идентификатор сессии и ссылку на механизм хеширования. Затем размер буфера хешированных данных можно определить, вызвав `C_Digest()`, и выполнить хеширование данных, вызвав `C_Digest()` второй раз.

Предварительно должна быть открыта сессия чтения/записи.

### Пример хеширования данных по алгоритму ГОСТ Р 34.11-94 и ГОСТ Р 34.11-2012

#### Хеширование данных по алгоритму ГОСТ Р 34.11-94 / ГОСТ Р 34.11-2012

```

/*      */
CK_BYTE pbtData[] = { 0x3C, 0x21, 0x50, 0x49, 0x4E, 0x50, 0x41, 0x44, 0x46, 0x49, 0x4C, 0x45, 0x20, 0x52, 0x55,
0x3E,
                                0x3C, 0x21, 0x3E, 0xED, 0xE5, 0xE2, 0xE8, 0xE4, 0xE8, 0xEC, 0xFB,
0xE9, 0x20, 0xF2, 0xE5, 0xEA };

/*      34.11-94 */
CK_MECHANISM gostR3411_946HashMech = {CKM_GOSTR3411, NULL_PTR, 0};
/*      34.11-2012(256) */
CK_MECHANISM gostR3411_12_256HashMech = {CKM_GOSTR3411_12_256, NULL_PTR, 0};
/*      34.11-2012(512) */
CK_MECHANISM gostR3411_12_512HashMech = {CKM_GOSTR3411_12_512, NULL_PTR, 0 };

CK_BYTE_PTR pbtHash          = NULL_PTR;          //
CK_ULONG      ulHashSize      = 0;                //

while(TRUE)
{
    ...

    /*      */
    printf("C_DigestInit");
    rv = pFunctionList->C_DigestInit(hSession,          //
                                     &gostR3411_946HashMech );    //
:

// gostR3411_946HashMech, gostR3411_12_256HashMech  gostR3411_12_512HashMech
if (rv != CKR_OK)

```

```

    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      */
    printf("C_Digest step 1");
    rv = pFunctionList->C_Digest( hSession,
                                pbtData,
                                arraysize(pbtData),
                                pbtHash,
                                &ulHashSize);

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    pbtHash = (CK_BYTE*)malloc(ulHashSize);
    if (pbtHash == NULL)
    {
        printf("Memory allocation for pbtHash failed! \n");
        break;
    }
    memset(pbtHash,
           0,
           (ulHashSize * sizeof(CK_BYTE)));
    /*      */
    printf("C_Digest step 2");
    rv = pFunctionList->C_Digest(hSession,
                                pbtData,
                                arraysize(pbtData),
                                pbtHash,
                                &ulHashSize);

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
    break;
}

```

## Вычисление MAC (Message Authentication Code)

### Поддерживаемые механизмы

Устройства Рутокен поддерживает создание MAC на ключах типа СКК\_KUZNECHIK, СКК\_MAGMA и СКК\_GOST28147. Объекты ключей, на которых вычисляется и проверяется MAC, также должны иметь атрибуты SKA\_SIGN и SKA\_VERIFY установленные в SK\_TRUE.

### Вычисление MAC

Для вычисления MAC служат функции C\_SignInit() и C\_Sign(). Сначала операцию получения MAC нужно инициализировать через C\_SignInit(), передав в нее идентификатор сессии и ссылку на механизм. Размер буфера под MAC можно определить, вызвав C\_Sign() с нулевым указателем на выходные данные. Вызвав C\_Sign() с не нулевым указателем на выходные данные мы получим вычисленный MAC.

Предварительно должна быть открыта сессия чтения/записи.

### Пример получения MAC от данных по алгоритму ГОСТ Р 34.12-2015

## Получение MAC от данных по алгоритму ГОСТ Р 34.12-2015

```
/*      */
CK_BYTE pbtData[] = { 0x3C, 0x21, 0x50, 0x49, 0x4E, 0x50, 0x41, 0x44, 0x46, 0x49, 0x4C, 0x45, 0x20, 0x52, 0x55,
0x3E,
                                0x3C, 0x21, 0x3E, 0xED, 0xE5, 0xE2, 0xE8, 0xE4, 0xE8, 0xEC, 0xFB,
0xE9, 0x20, 0xF2, 0xE5, 0xEA };

/*      28147-89 */
CK_MECHANISM gost28147MacMech = {CKM_GOST28147_MAC, NULL_PTR, 0};
/*      34.12-2015 */
CK_MECHANISM kuznechikMacMech = { CKM_KUZNECHIK_MAC, NULL_PTR, 0 };
CK_MECHANISM magmaMacMech = { CKM_MAGMA_MAC, NULL_PTR, 0 };

CK_BYTE_PTR pbtMac      = NULL_PTR;          //      MAC
CK_ULONG      ulMacSize      = 0;          //

while(TRUE)
{
    ...

    /*      MAC*/
    printf("C_SignInit");
    rv = functionList->C_SignInit(hSession,
                                &kuznechikMacMech, //      //      MAC
                                hSecretKey); //      ,      MAC

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      MAC */
    printf("C_Sign step 1");
    rv = pFunctionList->C_Sign( hSession,
                                //
                                pbtData, //
                                //
                                arraysize(pbtData), //
                                pbtMac, //
                                &ulMacSize); //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    pbtMac = (CK_BYTE*)malloc(ulMacSize);
    if (pbtMac == NULL)
    {
        printf("Memory allocation for pbtMac failed! \n");
        break;
    }
    memset(pbtMac,
           0,
           (ulMacSize * sizeof(CK_BYTE)));
    /*      MAC      */
    printf("C_Sign step 2");
    rv = pFunctionList->C_Sign( hSession,
                                //
                                pbtData, //
                                arraysize(pbtData), //
                                pbtMac, //
                                &ulMacSize); //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
}
```

```
        break;
    }
```

## Проверка MAC

Для проверки MAC служат функции `C_VerifyInit()` и `C_Verify()`. Сначала операцию проверки MAC нужно инициализировать через `C_VerifyInit()`, передав в нее идентификатор сессии и ссылку на механизм. Проверка MAC происходит при вызове функции `C_Verify()` с переданным указателем на данные и MAC.

Предварительно должна быть открыта сессия чтения/записи.

### Пример проверки MAC от данных по алгоритму ГОСТ Р 34.12-2015

#### Проверка MAC от данных по алгоритму ГОСТ Р 34.12-2015

```
/*      */
CK_BYTE pbtData[] = { 0x3C, 0x21, 0x50, 0x49, 0x4E, 0x50, 0x41, 0x44, 0x46, 0x49, 0x4C, 0x45, 0x20, 0x52, 0x55,
0x3E,
                                0x3C, 0x21, 0x3E, 0xED, 0xE5, 0xE2, 0xE8, 0xE4, 0xE8, 0xEC, 0xFB,
0xE9, 0x20, 0xF2, 0xE5, 0xEA };

/* MAC 34.12-2015 */
CK_MECHANISM kuznechikMacMech = { CKM_KUZNECHIK_MAC, NULL_PTR, 0 };
CK_MECHANISM magmaMacMech = { CKM_MAGMA_MAC, NULL_PTR, 0 };

while(TRUE)
{
    ...

    /* MAC*/
    printf("C_VerifyInit");
    rv = functionList->C_VerifyInit(hSession,
                                    //
                                    &kuznechikMacMech, // MAC
                                    hSecretKey); //

    MAC
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /* MAC */
    printf("C_Verify");
    rv = pFunctionList->C_Verify( hSession,
                                    //
                                    pbtData, //
                                    //
                                    arraysize(pbtData), //
                                    pbtMac, //
                                    &ulMacSize); //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
    break;
}
```

## Подпись и проверка подписи

## Поддерживаемые механизмы

Устройства Рутокен поддерживают следующие механизмы подписи:

- СКМ\_GOSTR3410 подписи алгоритмом ГОСТ Р 34.10.2001 и ГОСТ Р 34.10.2012 с длиной закрытого ключа 256 бит,
- СКМ\_GOSTR3410\_WITH\_GOSTR3411 для совместного хеширования алгоритмом СКМ\_GOSTR3411 и подписи алгоритмом СКМ\_GOSTR3410,
- СКМ\_GOSTR3410\_512 для подписи алгоритмом ГОСТ Р 34.10.2012 с длиной закрытого ключа 512 бит,
- СКМ\_GOSTR3410\_WITH\_GOSTR3411\_12\_256 для совместного хеширования алгоритмом СКМ\_GOSTR3411\_12\_256 и подписи на ключе длиной 256 бит,
- СКМ\_GOSTR3410\_WITH\_GOSTR3411\_12\_512 для совместного хеширования алгоритмом СКМ\_GOSTR3411\_12\_512 и подписи на ключе длиной 512 бит,
- СКМ\_RSA\_PKCS для подписи алгоритмом RSA.
- СКМ\_MD5\_RSA\_PKCS, СКМ\_SHA1\_RSA\_PKCS, СКМ\_SHA224\_RSA\_PKCS, СКМ\_SHA256\_RSA\_PKCS, СКМ\_SHA384\_RSA\_PKCS, СКМ\_SHA512\_RSA\_PKCS для алгоритма взятия хеша с последующим взятием подписи по алгоритму RSA
- СКМ\_ECDSA для подписи алгоритмом ECDSA.

## Подпись данных

Для вычисления подписи сообщения служат функции `C_SignInit()` и `C_Sign()`. Сначала операцию подписи нужно инициализировать через `C_SignInit()`, передав в нее идентификатор сессии, механизма и закрытого ключа. Затем размер буфера для подписанных данных можно определить, вызвав `C_Sign()` с указателем на длину буфера, равной нулю, и подписать данные, вызвав `C_Sign()` второй раз.

Функция `C_Sign()`, как и некоторые другие функции интерфейса PKCS#11, возвращает значения различной длины и подчиняется соответствующему соглашению вызова для подобных функций, описанному в пункте 11.2 стандарта PKCS#11. Краткая его суть сводится к тому, что существует два способа вызова таких функций. В первом случае функция может быть вызвана с пустым указателем для возвращаемого буфера (`NULL_PTR`), в таком случае функция вернет размер возвращаемого буфера в байтах в соответствующую переменную. Во втором случае функция может быть вызвана с непустым указателем для возвращаемого буфера и указанным значением длины буфера, в таком случае функция вернет результат выполнения криптографической операции в соответствующую переменную при условии достаточности размера буфера. Если указанная длина буфера недостаточна для возвращения всего результат, функция вернет ошибку `CKR_BUFFER_TOO_SMALL`. При успешном выполнении функция возвращает код ошибки `CKR_OK`.

При использовании совместных алгоритмов хеширования и подписи (например, `СКМ_GOSTR3410_WITH_GOSTR3411`) в `C_Sign()` передается открытый текст для подписи, при использовании только алгоритма подписи (например, `СКМ_GOSTR3410`) – уже прохешированные данные.

При использовании совместных алгоритмов хеширования и подписи в механизме должны быть заданы параметры алгоритма хеширования.

В качестве данных на подпись может быть передан запрос на сертификат, представленный в байт-коде.

## Подпись данных отдельными механизмами хеширования и подписи

При использовании отдельных механизмов хеширования и подписи сообщение сначала хешируется функциями `C_DigestInit()` и `C_Digest()`, а затем значение хеша подписывается функциями `C_SignInit()` и `C_Sign()`.

## Пример подписи данных по алгоритму ГОСТ Р 34.10-2012 отдельными механизмами хеширования и подписи для всех устройств Рутокен

### Подпись данных по алгоритму ГОСТ Р 34.10-2012

```
/*          */
CK_BYTE pbtData[] = { 0x3C, 0x21, 0x50, 0x49, 0x4E, 0x50, 0x41, 0x44, 0x46, 0x49, 0x4C, 0x45, 0x20, 0x52, 0x55,
0x3E,
                                0x3C, 0x21, 0x3E, 0xED, 0xE5, 0xE2, 0xE8, 0xE4, 0xE8, 0xEC, 0xFB,
0xE9, 0x20, 0xF2, 0xE5, 0xEA,
                                0xF1, 0xF2, 0x3C, 0x4E, 0x3E, 0xD4, 0xC8, 0xCE, 0x3A, 0x3C, 0x56,
0x3E, 0xCF, 0xE5, 0xF2, 0xF0,
                                0xEE, 0xE2, 0x20, 0xCF, 0xE5, 0xF2, 0xF0, 0x20, 0xCF, 0xE5, 0xF2,
0xF0, 0xEE, 0xE2, 0xE8, 0xF7,
                                0x20, 0xCC, 0xEE, 0xF1, 0xEA, 0xE2, 0xE0, 0x2C, 0x20, 0xCF, 0xE8,
0xEE, 0xED, 0xE5, 0xF0, 0xF1,
                                0xEA, 0xE0, 0xFF, 0x20, 0xF3, 0xEB, 0x2C, 0x20, 0xE4, 0x2E, 0x20,
0x33, 0x2C, 0x20, 0xEA, 0xE2,
                                0x2E, 0x20, 0x37, 0x32 };

/*          34.11-2012(256) */
CK_MECHANISM HashMech256 = {СКМ_GOSTR3411_12_256, NULL_PTR, 0};

/*          34.11-2012(512) */
CK_MECHANISM HashMech512 = {СКМ_GOSTR3411_12_512, NULL_PTR, 0};
```

```

/* /      34.10-2012, 256 */
CK_MECHANISM    SigVerMech256 = {CKM_GOSTR3410, NULL_PTR, 0};

/* /      34.10-2012, 512 */
CK_MECHANISM    SigVerMech512 = {CKM_GOSTR3410_512, NULL_PTR, 0};

CK_BYTE_PTR    pbtHash          = NULL_PTR;      //
CK_ULONG       ulHashSize       = 0;            //

CK_BYTE_PTR    pbtSignature     = NULL_PTR;     // ,
CK_ULONG       ulSignatureSize = 0;            // , ,

while(TRUE)
{
    ...

    /*      */
    printf("C_DigestInit");
    rv = pFunctionList->C_DigestInit(hSession,          //
                                     &HashMech256);    //

    (HashMech256 HashMech512)
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      */
    printf("C_Digest step 1");
    rv = pFunctionList->C_Digest( hSession,          //
                                  pbtData,          //
                                  arraysize(pbtData), //
                                  pbtHash,          //
                                  &ulHashSize);    //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    pbtHash = (CK_BYTE*)malloc(ulHashSize);
    if (pbtHash == NULL_PTR)
    {
        printf("Memory allocation for pbtHash failed! \n");
        break;
    }
    memset(pbtHash,
           0,
           (ulHashSize * sizeof(CK_BYTE)));

    /*      */
    printf("C_Digest step 2");
    rv = pFunctionList->C_Digest(hSession,          //
                                  pbtData,          //
                                  arraysize(pbtData), //
                                  pbtHash,          //
                                  &ulHashSize);    //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      */
    printf("C_SignInit");

```

```

        rv = pFunctionList->C_SignInit( hSession,                //
                                        &SigVerMech256,          //
(SigVerMech256  SigVerMech512)
                                        hPrivateKey );          //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /* */
    printf("C_Sign step 1");
    rv = pFunctionList->C_Sign( hSession,                        //
                                pbtHash,                         //
                                ulHashSize,                     //
                                pbtSignature,                   //
                                &ulSignatureSize);             //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    pbtSignature = (CK_BYTE*)malloc(ulSignatureSize);
    if (pbtSignature == NULL)
    {
        printf("Memory allocation for pbtSignature failed! \n");
        break;
    }
    memset( pbtSignature,
            0,
            ulSignatureSize * sizeof(CK_BYTE));

    /* */
    printf("C_Sign step 2");
    rv = pFunctionList->C_Sign( hSession,                        //
                                pbHash,                         //
                                ulHashSize,                     //
                                pbtSignature,                   //
                                &ulSignatureSize);             //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /* , */
    printf("Signature buffer is: \n");
    for (i = 0;
         i < ulSignatureSize;
         i++)
    {
        printf("%02X ", pbtSignature[i]);
        if ((i + 1) % 8 == 0)
            printf("\n");
    }

    break;
}

```

### Подпись совместным механизмом хеширования и подписи

При использовании совместного механизма и хеширование, и подпись выполняются функцией C\_Sign(). Сначала в функцию C\_SignInit() передается совместный механизм (например, СКМ\_GOSTR3410\_WITH\_GOSTR3411), а затем в функцию C\_Sign() – сообщение.

### Пример подписи данных по алгоритму ГОСТ Р 34.10-2012 совместным механизмом хеширования и подписи



## Подпись данных по алгоритму ГОСТ Р 34.10-2012

```
/*      */
CK_BYTE pbtData[] = { 0x3C, 0x21, 0x50, 0x49, 0x4E, 0x50, 0x41, 0x44, 0x46, 0x49, 0x4C, 0x45, 0x20, 0x52, 0x55,
0x3E,
                                0x3C, 0x21, 0x3E, 0xED, 0xE5, 0xE2, 0xE8, 0xE4, 0xE8, 0xEC, 0xFB,
0xE9, 0x20, 0xF2, 0xE5, 0xEA,
                                0xF1, 0xF2, 0x3C, 0x4E, 0x3E, 0xD4, 0xC8, 0xCE, 0x3A, 0x3C, 0x56,
0x3E, 0xCF, 0xE5, 0xF2, 0xF0,
                                0xEE, 0xE2, 0x20, 0xCF, 0xE5, 0xF2, 0xF0, 0x20, 0xCF, 0xE5, 0xF2,
0xF0, 0xEE, 0xE2, 0xE8, 0xF7,
                                0x20, 0xCC, 0xEE, 0xF1, 0xEA, 0xE2, 0xE0, 0x2C, 0x20, 0xCF, 0xE8,
0xEE, 0xED, 0xE5, 0xF0, 0xF1,
                                0xEA, 0xE0, 0xFF, 0x20, 0xF3, 0xEB, 0x2C, 0x20, 0xE4, 0x2E, 0x20,
0x33, 0x2C, 0x20, 0xEA, 0xE2,
                                0x2E, 0x20, 0x37, 0x32 };

/*      34.11-2012, 256 */
CK_BYTE GOST3411_256_params[] = { 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x02 };

/*      34.10-2012      34.11-2012 (256) */
CK_MECHANISM HashSigVerMech256 = {CKM_GOSTR3410_WITH_GOSTR3411_12_256, GOST3411_256_params, sizeof
(GOST3411_256_params)};

/*      34.11-2012, 512 */
CK_BYTE GOST3411_512_params[] = { 0x06, 0x08, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x01, 0x02, 0x03 };

/*      34.10-2012      34.11-2012 (512) */
CK_MECHANISM HashSigVerMech512 = {CKM_GOSTR3410_WITH_GOSTR3411_12_512, GOST3411_512_params, sizeof
(GOST3411_512_params)};

CK_BYTE_PTR pbtSignature = NULL_PTR;          // ,
CK_ULONG ulSignatureSize = 0;                // ,

while(TRUE)
{
    ...

    /*      */
    printf("C_SignInit");
    rv = pFunctionList->C_SignInit(hSession,          //
                                &HashSigVerMech256, //
                                (HashSigVerMech256 HashSigVerMech512)
                                hPrivateKey );      //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      */
    printf("C_Sign step 1");
    rv = pFunctionList->C_Sign(hSession,          //
                                pbtData,          //
                                arraysize(pbtData), //
                                pbtSignature,     //
                                &ulSignatureSize); //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    pbtSignature = (CK_BYTE*)malloc(ulSignatureSize);
    if (pbtSignature == NULL)
    {
```

```

        printf("Memory allocation for pbtSignature failed! \n");
        break;
    }
    memset( pbtSignature,
           0,
           ulSignatureSize * sizeof(CK_BYTE));

    /* */
    printf("C_Sign step 2");
    rv = pFunctionList->C_Sign(hSession,
                               //
                               pbtData, //
                               arraysize(pbtData), //
                               pbtSignature, //
                               &ulSignatureSize); //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /* , */
    printf("Signature buffer is: \n");
    for (i = 0;
         i < ulSignatureSize;
         i++)
    {
        printf("%02X ", pbtSignature[i]);
        if ((i + 1) % 8 == 0)
            printf("\n");
    }

    break;
}

```

## Пример подписи данных по алгоритму ECDSA отдельными механизмами хеширования и подписи

### Подпись данных по алгоритму ECDSA

```

/*****
*
*****/
CK_BYTE pbtData[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07 };

CK_MECHANISM sha256Mech = { CKM_SHA256, NULL_PTR, 0 };
CK_MECHANISM ecdsaSigVerMech = { CKM_ECDSA, NULL_PTR, 0 };

CK_BYTE_PTR pbtSignature; // ,
CK_ULONG ulSignatureSize ; // ,

CK_BYTE_PTR pbtHash ; // -
CK_ULONG ulHashSize; //

while (TRUE)
{
    ...

    /* */
    rv = pFunctionList->C_DigestInit(hSession, &sha256Mech);
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
}

```

```

}
printf(" -> OK\n");

/*      */
printf("C_Digest step 1");
rv = pFunctionList->C_Digest(hSession, pbtData, sizeof(pbtData), NULL_PTR, &ulHashSize);

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

pbtHash = (CK_BYTE*)malloc(ulHashSize);
if (pbtHash == NULL_PTR)
{
    printf("Memory allocation for pbtHash failed! \n");
    break;
}
memset(pbtHash,
        0,
        (ulHashSize * sizeof(CK_BYTE)));
/*      */
printf("C_Digest step 2");
rv = pFunctionList->C_Digest(hSession, pbtData, sizeof(pbtData), pbtHash, &ulHashSize);
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

/*      */
printf("C_SignInit");
rv = pFunctionList->C_SignInit(hSession, &ecdsaSigVerMech, hPrivateKey);
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

pbtSignature = (CK_BYTE*)malloc(ulSignatureSize);
if (pbtSignature == NULL_PTR)
{
    printf("Memory allocation for pbtSignature failed! \n");
    break;
}
memset(pbtSignature,
        0,
        ulSignatureSize * sizeof(CK_BYTE));

/*      */
printf("C_Sign step 2");
rv = pFunctionList->C_Sign(hSession, pbtHash, ulHashSize, NULL_PTR, &ulSignatureSize);
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

/*      */
printf("Signature buffer is: \n");
for (i = 0;
     i < ulSignatureSize;
     i++)
{
    printf("%02X ", pbtSignature[i]);
    if ((i + 1) % 8 == 0)

```

```

        printf("\n");
    }
    break;
}

```

## Проверка подписи

Для проверки подписи данных служат функции `C_VerifyInit()` и `C_Verify()`. Сначала операцию подписи нужно инициализировать через `C_VerifyInit()`, передав в нее идентификатор сессии, механизма и открытого ключа. Затем можно проверить подпись функцией `C_Verify()`.

Совместные механизмы хеширования и подписи (например, `СКМ_GOSTR3410_WITH_GOSTR3411`) не поддерживаются `C_VerifyInit()`. `C_Verify()` передаются предварительно прохешированные функцией `C_Digest()` исходные данные.

### Проверка подписи

```

while(TRUE)
{
    ...
    /*      */
    printf(" C_VerifyInit");
    rv = pFunctionList->C_VerifyInit(hSession,          //
                                    &SigVerMech,      //
                                    hPublicKey);       //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      */
    printf(" C_Verify");
    rv = pFunctionList->C_Verify(hSession,             //
                                pbHash,                //
                                ulHashSize,           //
                                pbtSignature,         //
                                ulSignatureSize);     //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
    break;
}
...
if (pbtSignature)
{
    free(pbtSignature);
    pbtSignature = NULL_PTR;
}

if (pbHash)
{
    free(pbHash);
    pbHash = NULL_PTR;
}

```

## Поточная подпись и проверка подписи

При работе с большим количеством данных бывает удобно отправлять данные токену частями. При таком режиме работе с токеном необходимо использовать функции `C_SignInit/C_VerifyInit`, `C_SignUpdate/C_VerifyUpdate` и `C_SignFinal/C_VerifyFinal`.

## Пример подписи данных в несколько итераций

```
/* */
printf("C_SignInit");
rv = pFunctionList->C_SignInit( hSession,          //
                                &SigVerMech256,   //
(SigVerMech256 SigVerMech512)
                                hPrivateKey );     //

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

for (size_t i=0; i < dataLen; i+=blockLen)
{
    size_t len = (dataLen-i) < blockLen? (dataLen-i) : blockLen;
    //
    rv = pFunctionList->C_SignUpdate( hSession,    //
i,                                           //
len);                                       //

    if (rv != CKR_OK)
        break;
}

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

/* */
printf("C_SignFinal step 1");
rv = pFunctionList->C_SignFinal(hSession,      //
                                pbtSignature,   //
                                &ulSignatureSize); //

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

pbtSignature = (CK_BYTE*)malloc(ulSignatureSize);
if (pbtSignature == NULL)
{
    printf("Memory allocation for pbtSignature failed! \n");
    break;
}
memset( pbtSignature,
        0,
        ulSignatureSize * sizeof(CK_BYTE));

/* */
printf("C_SignFinal step 2");
rv = pFunctionList->C_SignFinal(hSession,      //
                                pbtSignature,   //
                                &ulSignatureSize); //

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");
```

# Шифрование и расшифрование

## Поддерживаемые механизмы

Устройства Рутокен поддерживают следующие механизмы шифрования:

- СКМ\_GOST28147\_ECB для шифрования алгоритмом ГОСТ 28147-89 в режиме простой замены,
- СКМ\_GOST28147 для шифрования алгоритмом ГОСТ 28147-89 в режиме гаммирования с обратной связью (программное и аппаратное),
- СКМ\_KUZNECHIK\_ECB для шифрования алгоритмом Кузнечик в режиме простой замены (ГОСТ 34.13-2018),
- СКМ\_KUZNECHIK\_CTR\_ACPKM для шифрования алгоритмом Кузнечик в режиме гаммирования CTR с мешингом АСРКМ (Р 1323565.1.017-2018),
- СКМ\_MAGMA\_ECB для шифрования алгоритмом Магма в режиме простой замены (ГОСТ 34.13-2018),
- СКМ\_MAGMA\_CTR\_ACPKM для шифрования алгоритмом Магма в режиме гаммирования CTR с мешингом АСРКМ (Р 1323565.1.017-2018),
- СКМ\_RSA\_PKCS для шифрования алгоритмом RSA.

Внимание!



Так как в режиме простой замены (механизм СКМ\_GOST28147\_ECB) шифрование каждого блока данных осуществляется одним и тем же ключом, этот механизм должен применяться только для данных небольшого размера (например, ключей). В противном случае стойкость алгоритма снижается.

Для того, чтобы шифрование/расшифрование было выполнено аппаратно самим устройством, ключ должен находиться на токене (то есть атрибут СКА\_ТОКЕН используемого для шифрования ключа должен быть равен значению TRUE). Если атрибут СКА\_ТОКЕН ключа, который используется для шифрования/расшифрования, равен FALSE, то операция будет выполняться программно.

## Шифрование данных

Для шифрования данных одним блоком служат функции C\_EncryptInit() и C\_Encrypt(), для поточного – \_EncryptInit(), C\_EncryptUpdate() и C\_EncryptFinal().

Сначала операцию шифрования нужно инициализировать вызовом функции C\_EncryptInit(), передав в нее идентификатор сессии, механизма и секретного ключа. В параметрах механизмов для ГОСТ 28147-89 и ГОСТ Р 34.12-2015 можно задать вектор инициализации и его длину.

Далее шифрование можно выполнить для всего блока данных целиком, вызвав функцию C\_Encrypt(), или по частям, вызывая C\_EncryptUpdate() для каждого непоследнего блока и C\_EncryptFinal() для завершающего блока. Если в C\_EncryptFinal() передать пустой блок данных, то функция вернет суммарный размер всех блоков зашифрованных данных.

## Пример шифрования данных по алгоритмам ГОСТ 28147-89 и ГОСТ Р 34.12-2015

### Шифрование данных по алгоритмам ГОСТ 28147-89 ГОСТ Р 34.12-2015 в режиме простой замены

```
/* */
CK_BYTE pbtData[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
                      0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
                      0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09,
                      0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x00 };

/* */
CK_MECHANISM EncDecMech =
    { СКМ_GOST28147_ECB, NULL_PTR, 0 }; // 28147-89
// { СКМ_KUZNECHIK_ECB, NULL_PTR, 0 }; //
// { СКМ_MAGMA_ECB, NULL_PTR, 0 }; //

CK_BYTE_PTR pbtEncryptedData = NULL_PTR; // ,
CK_ULONG ulEncryptedDataSize = 0; // ,

while(TRUE)
{
    ...
    /* */
    printf("C_EncryptInit");
    rv = pFunctionList->C_EncryptInit(hSession, //
```

```

                                &EncDecMech,          //
                                hSecKey);              //
CKA_TOKEN TRUE,                ,                      //
.                               ,                      //
                                // -

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    /*      */
    printf("Getting encrypted data size");
    rv = pFunctionList->C_Encrypt(hSession,           //
                                pbtData,             //
                                arraysize(pbtData),  //
                                NULL,                //
                                &ulEncryptedDataSize); //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    pbtEncryptedData = (CK_BYTE*)malloc(ulEncryptedDataSize);
    if (pbtEncryptedData == NULL)
    {
        printf("Memory allocation for pbtEncryptedData failed! \n");
        break;
    }
    memset(pbtEncryptedData,
           0,
           (ulEncryptedDataSize * sizeof(CK_BYTE)));

    /*      */
    printf("C_Encrypt");
    rv = pFunctionList->C_Encrypt(hSession,          //
                                pbtData,             //
                                arraysize(pbtData),  //
                                pbtEncryptedData,    //
                                &ulEncryptedDataSize); //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    printf("Encrypted buffer is:\n");
    for (i = 0;
         i < ulEncryptedDataSize;
         i++)
    {
        printf("%02X ", pbtEncryptedData[i]);
        if ((i + 1) % 8 == 0)
            printf("\n");
    }
    break;
}

```

Шифрование данных по алгоритмам ГОСТ 28147-89 в режиме гаммирования с обратной связью и ГОСТ Р 34.12-2015 в режиме гаммирования CTR с мешингом АСРКМ

```
/*      */
```





```

ulBlockSize, // , pbtEncryptedData +
ulCurrentPosition, //
&ulBlockSize); // ,
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");

    ulCurrentPosition += ulBlockSize;
    ulRestLen -= ulBlockSize;
}
if (rv != CKR_OK)
    break;

printf("Finalizing encryption");
rv = pFunctionList->C_EncryptFinal( hSession, //
NULL_PTR, //
&ulEncryptedDataSize); //
    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
    /* , */
    printf("Encrypted buffer is:\n");
    for (i = 0;
        i < ulEncryptedDataSize;
        i++)
    {
        printf("%02X ", pbtEncryptedData[i]);
        if ((i + 1) % 8 == 0)
            printf("\n");
    }

    break;
}

```

## Расшифрование данных

Для расшифрования данных служат функции `C_DecryptInit()` и `C_Decrypt()` для любого режима шифрования.

Сначала операцию расшифрования нужно инициализировать вызовом функции `C_DecryptInit()`, передав в нее идентификатор сессии, механизма и секретного ключа. В параметрах механизма для ГОСТ 28147-89 можно задать вектор инициализации и его длину.

Далее расшифрование выполняется вызовом функции `C_Decrypt()` с передачей в нее зашифрованные данные. Размер расшифрованных данных можно узнать, вызвав `C_Decrypt()` с пустым указателем вместо указателя на буфера для расшифрованных данных.

## Пример расшифрования данных по алгоритму ГОСТ 28147-89

### Расшифрование данных по алгоритму ГОСТ 28147-89

```

while(TRUE)
{
    ...
    /* */
    printf("C_DecryptInit");
    rv = pFunctionList->C_DecryptInit(hSession, //
                                     &EncDecStreamMech, //
                                     hSecKey); //
}

```

```

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

/* */
printf("Getting decrypted data size");
rv = pFunctionList->C_Decrypt(hSession,
                               pbtEncryptedData,
                               ulEncryptedDataSize,
                               NULL_PTR,
                               &ulDecryptedDataSize);

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");
pbtDecryptedData = (CK_BYTE*)malloc(ulDecryptedDataSize);
if (pbtDecryptedData == NULL)
{
    printf("Memory allocation for pbtDecryptedData failed! \n");
    break;
}
memset(pbtDecryptedData,
        0,
        (ulDecryptedDataSize * sizeof(CK_BYTE)));

printf("C_Decrypt");
rv = pFunctionList->C_Decrypt(hSession,
                               pbtEncryptedData,
                               ulEncryptedDataSize,
                               pbtDecryptedData,
                               &ulDecryptedDataSize);

if (rv != CKR_OK)
{
    printf(" -> Failed\n");
    break;
}
printf(" -> OK\n");

/* , */
printf("Decrypted buffer is:\n");
for (i = 0;
     i < ulDecryptedDataSize;
     i++)
{
    printf("%02X ", pbtDecryptedData[i]);
    if ((i + 1) % 8 == 0)
        printf("\n");
}
break;
}

if (pbtEncryptedData)
{
    free(pbtEncryptedData);
    pbtEncryptedData = NULL_PTR;
}
if (pbtDecryptedData)
{
    free(pbtDecryptedData);
    pbtDecryptedData = NULL_PTR;
}

```

## Форматирование токена

Форматирование токена возможно двумя функциями: функцией расширения `C_EX_InitToken()`, которая полностью очищает память токена и сбрасывает все настройки и PIN-коды, и стандартной функцией `C_InitToken()`, которая удаляет только объекты PKCS#11.

Все параметры форматирования задаются структурой типа `CK_RUTOKEN_INIT_PARAM`, указатель на которую вместе с PIN-кодом Администратора передаются функции `C_EX_InitToken()`.

### Форматирование Рутокен функцией `C_EX_InitToken()`

```
/* PIN- */
#define MAX_ADMIN_RETRY_COUNT      10
/* */
#define MAX_USER_RETRY_COUNT      10
/* DEMO PIN- */
static CK_UTF8CHAR      NEW_USER_PIN[] = {'5', '5', '5', '5', '5', '5', '5', '5'};
/* DEMO PIN- */
static CK_UTF8CHAR      SO_PIN[]      = {'8', '7', '6', '5', '4', '3', '2', '1'};
/* DEMO */
static CK_CHAR TOKEN_STD_LABEL[]     = {"!!!Sample Rutoken label!!!"};

CK_RUTOKEN_INIT_PARAM initInfo_st;      // CK_RUTOKEN_INIT_PARAM, C_EX_InitToken
CK_BBOOL bIsRutokenECP = FALSE;        //

/* CK_RUTOKEN_INIT_PARAM */
memset(&initInfo_st,
      0,
      sizeof(CK_RUTOKEN_INIT_PARAM));

initInfo_st.ulSizeofThisStructure = sizeof(CK_RUTOKEN_INIT_PARAM); //
initInfo_st.UseRepairMode =
0; // 0
- , !0 -
initInfo_st.pNewAdminPin =
SO_PIN; // PIN-
initInfo_st.ulNewAdminPinLen = sizeof
(SO_PIN); // PIN-
initInfo_st.pNewUserPin =
NEW_USER_PIN; // PIN-
initInfo_st.ulNewUserPinLen = sizeof
(NEW_USER_PIN); // PIN-
initInfo_st.ulMinAdminPinLen = bIsRutokenECP ? 6 : 1; //
PIN-
initInfo_st.ulMinUserPinLen = bIsRutokenECP ? 6 : 1; //
PIN-
initInfo_st.ChangeUserPINPolicy
= // PIN- -
(TOKEN_FLAGS_ADMIN_CHANGE_USER_PIN | TOKEN_FLAGS_USER_CHANGE_USER_PIN);
initInfo_st.ulMaxAdminRetryCount = MAX_ADMIN_RETRY_COUNT; //

initInfo_st.ulMaxUserRetryCount = MAX_USER_RETRY_COUNT;
//
initInfo_st.pTokenLabel =
TOKEN_STD_LABEL; //
initInfo_st.ulLabelLen = sizeof
(TOKEN_STD_LABEL); //

/* */
printf("Initializing token");
rv = pFunctionListEx->C_EX_InitToken(aSlots[0], //
SO_PIN, // PIN-
arraysize(SO_PIN), // PIN-
&initInfo_st); //

if (rv != CKR_OK)
printf(" -> Failed\n");
else
printf(" -> OK\n");
```

При форматировании стандартной функцией `C_InitToken()` возможно задать только новый PIN-код Администратора и метку токена. После форматирования следует задать PIN-код Пользователя функцией `C_InitPIN()`, поскольку после форматирования через `C_InitToken()` его значение остается незадаанным.

#### Форматирование Рутокен функцией `C_InitToken()`

```
/* DEMO Rutoken */
static CK_CHAR TOKEN_LABEL[] = { 'M', 'Y', ' ', 'R', 'u', 't', 'o', 'k',
                                  'e', 'n', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
                                  ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
                                  ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' };

while(TRUE)
{
    ...

    /* */
    printf("C_InitToken");
    rv = pFunctionList->C_InitToken(aSlots[0],          //
                                    SO_PIN,              // PIN-
                                    sizeof(SO_PIN),      // PIN-
                                    TOKEN_LABEL);        //

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
    ...

    /* PIN- */
    printf(" Initializing User PIN ");
    rv = pFunctionList->C_InitPIN(hSession,             //
                                  USER_PIN,            // PIN-
                                  sizeof(USER_PIN));    // PIN-

    if (rv != CKR_OK)
    {
        printf(" -> Failed\n");
        break;
    }
    printf(" -> OK\n");
    break;
}
```

## Установка и смена локального PIN-кода

Помимо двух глобальных ролей Администратора и Пользователя, устройства Рутокен поддерживают до 29 PIN-кодов Локальных Пользователей для разных технических нужд.

Установка или смена локального PIN-кода должна выполняться Пользователем Рутокен и не требует открытой сессии и авторизации – PIN-код Пользователя передается прямо в функцию `C_EX_SetLocalPin()`.

### Установка локального PIN-кода

```
CK_SLOT_ID          slotID;                // ,
CK_UTF8CHAR_PTR    pUserPin;              // PIN-
CK_ULONG           ulUserPinLen;          // PIN-
CK_UTF8CHAR_PTR    pNewLocalPin;         // PIN-
CK_ULONG           ulNewLocalPinLen;      // PIN-
CK_ULONG           ulLocalID;            // PIN-

printf("Setting Local PIN-code");
rv = pfGetFunctionListEx->C_EX_SetLocalPIN( slotID,                // ,
                                             pUserPin,                //
                                             ulUserPinLen,            // PIN-
                                             pNewLocalPin,            // PIN-
                                             &ulNewLocalPinLen,        // PIN-
                                             ulLocalID);              // PIN-
if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
```

### Разблокировка PIN-кода Пользователя

В целях безопасности для пользовательского PIN-кода введен счетчик попыток неправильного ввода. При превышении заданного максимума попыток PIN-код блокируется и дальнейшая аутентификация с правами Пользователя становится невозможной. Сброс счетчика попыток неудачного входа осуществляется функцией `C_EX_UnblockUserPIN()`, в которую передается хэндл сессии с предварительно выполненным входом с правами Администратора.

Счетчик автоматически сбрасывается при вводе правильного PIN-кода Пользователя, если не был превышен заданный максимум попыток.

### Разблокировка токена

```
/* PIN- */
printf("Unlock User PIN");
rv = pFunctionListEx->C_EX_UnblockUserPIN(hSession); //
if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
```

## Управление памятью Рутокен ЭЦП Flash

### Получение объема флеш-памяти

Получить весь объем внешней флеш-памяти можно с помощью функции расширения `C_EX_GetDriveSize()`, передав в нее идентификатор слота с подключенным токеном и указатель на буфер, в который будет возвращен полученный объем памяти в Мб.

## Получение объема флэш-памяти Рутокен ЭЦП Flash

```
CK_ULONG ulDriveSize = 0; // -

printf("Get Flash memory size");
rv = pFunctionListEx->C_EX_GetDriveSize(aSlots[0], //
                                        &ulDriveSize); // -

if (rv != CKR_OK)
    printf(" -> Failed\n");
else
{
    printf(" -> OK\n");
    printf("Memory size: %d Mb\n", (int)ulDriveSize);
}
```

## Создание разделов флэш-памяти

Флэш-память Рутокен ЭЦП Flash может быть разбита на несколько независимых разделов с разными правами доступа к ним. Минимальное количество разделов – 1, максимальное – 8.

Рутокен ЭЦП Flash поддерживает следующие права доступа к разделам: для чтения и записи (ACCESS\_MODE\_RW), только для чтения (ACCESS\_MODE\_READ\_ONLY), скрытый раздел, защищенный от отображения в операционной системе, чтения, записи и любого другого типа доступа (ACCESS\_MODE\_HIDDEN) и раздел, эмулирующий CD-ROM (ACCESS\_MODE\_CD).

Владельцем раздела может выступать Администратор Рутокен (CKU\_SO), Пользователь Рутокен (CKU\_USER), а также локальный пользователь (с идентификатором в пределах от 0x03 до 0x1E) с предварительно заданным функцией C\_EX\_SetLocalPin() PIN-кодом.

Для разметки флэш-памяти на разделы предназначена функция C\_EX\_FormatDrive(). Вся информация о разделах (объем памяти, права доступа, владелец и флаги) задается в массиве структур типа CK\_VOLUME\_FORMAT\_INFO\_EXTENDED, который затем вместе с PIN-кодом Администратора и идентификатором слота, к которому подключен Рутокен, передается в функцию C\_EX\_FormatDrive().

## Создание разделов флеш-памяти Рутокен ЭЦП Flash

```
CK_ULONG          VolumeRWSize = 0;          //
CK_ULONG          VolumeROSize = 0;          //
CK_ULONG          VolumeCDSize = 0;          //   CD-ROM
CK_ULONG          VolumeHISize = 0;          //

CK_ULONG          CKU_LOCAL_1 = 0x03;        //
CK_ULONG          CKU_LOCAL_2 = 0x1E;        //

/*      */
CK_VOLUME_FORMAT_INFO_EXTENDED InitParams[] =
{
    { VolumeRWSize, ACCESS_MODE_RW, CKU_USER, 0 },
    { VolumeROSize, ACCESS_MODE_RO, CKU_SO, 0 },
    { VolumeHISize, ACCESS_MODE_HIDDEN, CKU_LOCAL_1, 0 },
    { VolumeCDSize, ACCESS_MODE_CD, CKU_LOCAL_2, 0 }
};

...

InitParams[0].ulVolumeSize = ulDriveSize / 2;
InitParams[1].ulVolumeSize = ulDriveSize / 4;
InitParams[2].ulVolumeSize = ulDriveSize / 8;
InitParams[3].ulVolumeSize = ulDriveSize - (ulDriveSize / 2) - (ulDriveSize / 4) - (ulDriveSize / 8);

printf("\nFormatting flash memory");
rv = pFunctionListEx->C_EX_FormatDrive( aSlots[0], //
CKU_SO, //
SO_PIN, //   PIN-
(SO_PIN), //   PIN-
InitParams, //
//   sizeof
//   arraysize(InitParams)); //
if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
```

## Получение информации о разделах флеш-памяти

Получить информацию о существующих на флеш-памяти разделах можно с помощью функции `C_EX_GetVolumesInfo()`, передав в нее идентификатор слота, к которому подключен токен, указатель на буфер и его размер, куда будет возвращен массив структур типа `CK_VOLUME_INFO_EXTENDED` с информацией о разделах (идентификатор раздела, его размер, права доступа, владелец и флаги). Размер буфера можно определить, вызвав `C_EX_GetVolumesInfo()` с пустым указателем на буфер.





## Изменение атрибутов разделов флеш-памяти Рутокен ЭЦП Flash

```
printf("\nChanging volume attributes");
rv = pFunctionListEx->C_EX_ChangeVolumeAttributes(aSlots[0],          //
CKU_SO,                      //
SO_PIN,                      // PIN-
(SO_PIN),                    // PIN-                                sizeof
VolumeRO,                    //
ACCESS_MODE_RW,             //
CK_TRUE);                   // CK_TRUE - , CK_FALSE -
if (rv != CKR_OK)
    printf(" -> Failed\n");
else
    printf(" -> OK\n");
```