Криптографический интерфейс ViPNet PKCS#11 VT 4.2

Руководство разработчика

infotecs

ФРКЕ.00106-04 33 02, версия 4.2.8

ОАО «ИнфоТеКС»

127287, г. Москва, Старый Петровско-Разумовский проезд, дом 1/23, строение 1

Тел: (495) 737-61-96 (горячая линия), 737-61-92, факс 737-72-78

Сайт компании «ИнфоТеКС»: http://infotecs.ru/

Электронный адрес службы поддержки: hotline@infotecs.ru

Содержание

Введение

О документе

Документ содержит описание интерфейса ViPNet PKCS#11, общую схему использования и функции для работы с интерфейсом.

Для кого предназначен документ

Документ предназначен для разработчиков программного обеспечения в области криптографического преобразования данных.

Соглашения документа

Ниже перечислены соглашения, принятые в этом документе для выделения информации.

Таблица 1. Обозначения, используемые в примечаниях

Обозначение	Описание
	**Внимание!** Указывает на обязательное для исполнения или следования действие или информацию.
	**Примечание.** Указывает на необязательное, но желательное для исполнения или следования действие или информацию.
	**Совет.** Содержит дополнительную информацию общего характера.

Таблица 2. Обозначения, используемые для выделения информации в тексте

Обозначение	Описание
Название	Название элемента интерфейса. Например, заголовок окна, название поля, кнопки или клавиши.
Клавиша+Клавиша	Сочетание клавиш. Чтобы использовать сочетание клавиш, следует нажать первую клавишу и, не отпуская ее, нажать вторую клавишу.
Меню > Подменю > Команда	Иерархическая последовательность элементов. Например, пункты меню или разделы на панели навигации.
`Код`	Имя файла, путь, фрагмент текстового файла (кода) или команда, выполняемая из командной строки.

Обратная связь

Дополнительная информация

Сведения о продуктах и решениях ViPNet, распространенные вопросы и другая полезная информация собраны на сайте ОАО «ИнфоТеКС»:

- Веб-портал документации ViPNet http://docs.infotecs.ru.
- Описание продуктов ViPNet https://infotecs.ru/product/.
- Информация о решениях ViPNet https://infotecs.ru/resheniya/.
- Сборник часто задаваемых вопросов (FAQ) https://infotecs.ru/support/faq/.
- Форум пользователей продуктов ViPNet https://infotecs.ru/forum/.

Контактная информация

С вопросами по использованию продуктов ViPNet, пожеланиями или предложениями свяжитесь со специалистами ОАО «ИнфоТеКС». Для решения возникающих проблем обратитесь в службу технической поддержки.

- Техническая поддержка для пользователей продуктов ViPNet: hotline@infotecs.ru.
- Форма запроса в службу технической поддержки https://infotecs.ru/support/request/.
- Консультации по телефону для клиентов, имеющих расширенный уровень технического сопровождения:

  8 (495) 737-6192,

  8 (800) 250-0260 — бесплатный звонок из любого региона России (кроме Москвы).

Распространение информации об уязвимостях продуктов ОАО «ИнфоТеКС» регулируется политикой ответственного разглашения https://infotecs.ru/disclosure.php. Если вы обнаружили уязвимости в продуктах компании, сообщите о них по адресу security-notifications@infotecs.ru.

Об интерфейсе ViPNet PKCS#11 VT

Назначение. Область применения

Криптографический интерфейс ViPNet PKCS#11 VT — один из криптографических интерфейсов, реализованных в продуктах ViPNet.

Криптографический интерфейс ViPNet PKCS#11 VT создан в соответствии с требованиями стандарта PKCS#11 v2.40: Cryptographic Token Interface Standard, разработанного в OASIS PKCS11 Technical Committee. Описание стандарта PKCS#11 v2.40 вы можете найти на сайте консорциума OASIS (https://docs.oasis-open.org/pkcs11/pkcs11-base/v2.40/pkcs11-base-v2.40.html). Интерфейс ViPNet PKCS#11 VT разработан для реализации в прикладных программах российских алгоритмов шифрования и электронной подписи.

Интерфейс описан в следующих заголовочных файлах:

- Стандартные функции PKCS#11:
  - Заголовочные файлы из PKCS#11 2.40:
    - `pkcs11.h`
    - `pkcs11f.h`
    - `pkcs11t.h`
  - Заголовочный файл с определениями, необходимыми для использования заголовочных файлов из стандарта:
    - `cryptoki.h`
  - Расширение стандартного интерфейса — объявления механизмов и атрибутов от Технического комитета по стандартизации «Криптографическая защита информации» (ТК 26):
    - `pkcs11tc26.h`
    - `pkcs11tc26_12.h`
    - `pkcs11tc26_15.h`
  - Расширение стандартного интерфейса — объявления механизмов и атрибутов, специфичных для ViPNet PKCS#11 VT:
    - `pkcs11ex.h`
- Расширение стандартного интерфейса — дополнительные функции, специфичные для ViPNet PKCS#11 VT:
  - `pkcs11_ex.h`
  - `pkcs11_extensions.h`
  - `pkcs11f_ex.h`

При использовании интерфейса ViPNet PKCS#11 VT в зависимости от необходимой функциональности должен быть включен один из следующих файлов:

- `cryptoki.h` — только стандартные функции и механизмы.
- `pkcs11tc26.h` — все перечисленное выше, а также механизмы от ТК26.
- `pkcs11ex.h` — все перечисленное выше, а также механизмы, специфичные для ViPNet PKCS#11 VT.

Если вам нужны дополнительные функции (см. «Функции расширения интерфейса PKCS#11» на стр. 99), то необходимо включить также файл `pkcs11_ex.h`.

Интерфейс ViPNet PKCS#11 VT реализован в виде динамической библиотеки `softtoken_pkcs11.dll` в ОС семейства Windows и `libsofttoken_pkcs11.so` в ОС Linux. В комплект поставки также входит исполняемый файл `token_manager`, который предназначен для создания виртуальных слотов и токенов для интерфейса ViPNet PKCS#11 VT.

С помощью интерфейса ViPNet PKCS#11 VT в прикладных программах можно выполнять следующие криптографические процедуры:

- Выработку секретных ключей для симметричного шифрования в соответствии с алгоритмом ГОСТ 28147-89.
- Шифрование и имитозащиту данных в соответствии с алгоритмом ГОСТ 28147-89.
- Выработку асимметричных ключей для осуществления и проверки электронной подписи в соответствии с алгоритмами ГОСТ Р 34.10-2001 и ГОСТ Р 34.10–2012.
- Выработку и проверку электронной подписи в соответствии с алгоритмами ГОСТ Р 34.10-2001 и ГОСТ Р 34.10–2012.
- Шифрование ключей ГОСТ 28147-89 с помощью ключей ГОСТ 28147-89.
- Хэширование данных в соответствии с алгоритмами ГОСТ Р 34.11-94 и ГОСТ Р 34.11–2012.
- Выработку общего секретного ключа ГОСТ 28147-89 при помощи ключей ГОСТ Р 34.10-2001.
- Выработку мастер-ключа, а также ключей шифрования и имитозащиты для TLS в соответствии с Российскими стандартами.
- Выработка секретного ключа из пароля с использованием PKCS#5v2 и хэш-функции ГОСТ Р 34.11-2012 256 бит.

Использование датчика случайных чисел

Датчик случайных чисел создает случайные последовательности чисел, на основе которых формируются закрытые ключи. ViPNet PKCS#11 VT использует настройки датчика ViPNet CSP (см. документ «ViPNet CSP. Руководство пользователя», раздел «Использование датчика случайных чисел»).

Названия и обозначения, применяемые в стандарте PKCS#11

Поскольку интерфейс ViPNet PKCS#11 VT создан в соответствии с требованиями стандарта PKCS#11 v2.20, в настоящем документе будет использоваться принятая в этом стандарте система обозначений, префиксов и определений, а именно:

Таблица 3. Символы

Символ	Определение
N/A	Not applicable
R/O	Read-only
R/W	Read/write

Таблица 4. Префиксы

Префикс	Описание
C_	Function
CK_	Data type or general constant
CKA_	Attribute
CKC_	Certificate type
CKD_	Key derivation function
CKF_	Bit flag
CKG_	Mask generation function
CKH_	Hardware feature type
CKK_	Key type
CKM_	Mechanism type
CKN_	Notification
CKO_	Object class
CKP_	Pseudo-random function
CKS_	Session state
CKR_	Return value

Префикс	Описание
CKU_	User type
CKZ_	Salt/Encoding parameter source
h	a handle
ul	a CK_ULONG
p	a pointer
pb	a pointer to a CK_BYTE
ph	a pointer to a handle
pul	a pointer to a CK_ULONG

Определения:

```
/* an unsigned 8-bit value */
typedef unsigned char CK_BYTE;
/* an unsigned 8-bit character */
typedef CK_BYTE CK_CHAR;
/* an 8-bit UTF-8 character */
typedef CK_BYTE CK_UTF8CHAR;
/* a BYTE-sized Boolean flag */
typedef CK_BYTE CK_BBOOL;
/* an unsigned value, at least 32 bits long */
typedef unsigned long int CK_ULONG;
/* a signed value, the same size as a CK_ULONG */
typedef long int CK_LONG;
/* at least 32 bits; each bit is a Boolean flag */
typedef CK_ULONG CK_FLAGS;
```

Указатели:

```
CK_BYTE_PTR   /* Pointer to a CK_BYTE */
CK_CHAR_PTR   /* Pointer to a CK_CHAR */
CK_UTF8CHAR_PTR /* Pointer to a CK_UTF8CHAR */
CK_ULONG_PTR   /* Pointer to a CK_ULONG */
CK_VOID_PTR   /* Pointer to a void */
CK_VOID_PTR_PTR /* Pointer to a CK_VOID_PTR */
NULL_PTR     /* A NULL pointer */
```

Во многих местах настоящего документа будет использоваться так называемое «соглашение МНД», то есть соглашение об использовании массивов неопределенной длины.

1

Слот и токен в ViPNet PKCS#11 VT

Общая информация

Основополагающими понятиями интерфейса PKCS#11 являются «слот» и «токен». Для «классических» реализаций интерфейса PKCS#11 слотом является, обычно, считыватель смарт-карт, а токеном — сама смарт-карта. Таким образом, токен является хранилищем некоторой персональной информации (различных ключей, сертификатов, приватных данных и тому подобное), а слот выступает в роли связующего звена между компьютером и токеном, допускающим подключение различных токенов в различное время.

В ViPNet PKCS#11 VT смарт-карты не поддерживаются, поэтому в роли слота и токена выступает «виртуальный токен», то есть некоторый файл из базы данных. В ОС Windows такие файлы находятся в рабочей области текущего пользователя (`%localappdata%\.softtoken_pkcs11\tokens`), в ОС Linux такие файлы находятся в каталоге `~/.itcs/soft-token/tokens`.

Кроме того, в ОС Windows в папке `%localappdata%\.softtoken_pkcs11`, а в ОС Linux — в каталоге `~/.itcs/soft-token` или `/opt/.itcs/soft-token` находится файл `settings.ini`, пример которого представлен ниже:

```
token_directory=C:\Users\%Username%\AppData\Local\.softtoken_pkcs11\tokens
db_perf=1
logging_enabled=true
logging_severity=3
object_caching_type=1
rng_config_file_path=
```

Остановимся на параметре, который может понадобиться вам в работе — это `logging_enabled` (по умолчанию включен). Если необходимо вести журналы работы модуля PKCS#11, этот параметр необходимо включить (присвоить значение `true`). По умолчанию в журнал заносятся только ошибки. Если требуется более подробное логирование, то необходимо сменить значение `logging_severity` на 6 или 7. В ОС Windows события записываются в `Debug Output`, в ОС Linux — в `syslog`).

Создание и удаление слотов и токенов в ViPNet PKCS#11 VT

Создание и удаление виртуальных слотов и токенов в ViPNet PKCS#11 VT, а также подключение и отключение токенов от слотов осуществляется с помощью прикладного модуля `token_manager`. Модуль может запускаться со следующими командами:

Операции:

- `--create_token` — создание токена `pkcs11`;
- `--remove_token` — удаление токена `pkcs11`;
- `--enum_tokens` — перечисление токенов `pkcs11`;
- `--display_token "token_id"` — отображение информации о выбранном токене. `token_id` — десятичное число.

Параметры:

- `--dll "dll_path"` — путь к библиотеке `pkcs11`;
- `--id "token_id"` — ID токена;
- `--label "label_string"` — метка токена;
- `--soPin "pin"` — ПИН-код администратора безопасности;
- `--userPin "pin"` — ПИН-код пользователя.

Пример запуска модуля:

```
tokenManager --dll softtoken_pkcs11.dll --createToken --id 10 --soPin 123456 --
userPin 123456
```

В результате создается файл токена `10.db` (в ОС Windows — в папке `%localappdata%\.softtoken_pkcs11\tokens`, в ОС Linux — в каталоге `~/.itcs/soft-token/tokens`) с ПИН-кодом администратора безопасности `123456` и ПИН-кодом пользователя `123456`.

2

Общая схема использования интерфейса PKCS#11

Пользователь и администратор

Поскольку токен является хранилищем персональных данных, то и доступ к этим данным должен быть персонифицирован и защищен от возможности проникновения к ним посторонних лиц. Поэтому в интерфейсе PKCS#11 используются такие понятия, как «пользователь» и «администратор» токена. Заметим, что администратор в документации OASIS именуется как Security Officer (SO).

Администратор выполняет следующие функции:

- инициализация и реинициализация токена;
- установка во вновь созданном токене ПИН-кода пользователя;
- смена ПИН-кода пользователя в том случае, если он забыл свой ПИН-код.

В интерфейсе PKCS#11 для администратора не предусмотрены прямые функции для записи или чтения приватной информации пользователя, хотя косвенно это всегда возможно: установить любой код в качестве ПИН-кода пользователя и прочесть всю приватную информацию. Поскольку создание «нового» токена в ViPNet PKCS#11 VT осуществляется с помощью модуля `token_manager.exe`, то основной функцией администратора токена является установление первоначального ПИН-кода пользователя и восстановление доступа к приватной информации пользователя в случае, если он забыл свой ПИН-код.

Атрибуты и объекты

Вся содержащаяся в токене информация представлена в виде объектов, а каждый объект состоит из набора атрибутов. Атрибут — это следующая структура:

```
typedef struct CK_ATTRIBUTE {
    CK_ATTRIBUTE_TYPE type;
    CK_VOID_PTR pValue;
    CK_ULONG ulValueLen;
} CK_ATTRIBUTE;
```

Где:

- `type` — тип атрибута (`typedef CK_ULONG CK_ATTRIBUTE_TYPE`);
- `pValue` — указатель на значение атрибута;
- `ulValueLen` — размер значения атрибута в байтах.

В ViPNet PKCS#11 VT используются стандартные атрибуты, определенные в `pkcs11t.h`, а также расширения из `pkcs11tc26.h` и `pkcs11ex.h`. Следует отметить, что в интерфейсе PKCS#11 могут быть два принципиально разных вида объектов: session- и token-объекты. Session-объекты — это объекты, существующие только в оперативной памяти компьютера во время сеанса связи с токеном (сессии), а token-объекты — это файлы, записанные в базу данных токена. Объект считается session-объектом, если среди его атрибутов нет атрибута `CKA_TOKEN` или же его значение равно `CK_FALSE`, в противном случае объект является token-объектом и записывается в токен.

Каждый объект в интерфейсе PKCS#11 должен иметь атрибут `CKA_CLASS`, определяющий класс объекта. В ViPNet PKCS#11 VT поддерживаются следующие классы объектов (`typedef CK_ULONG CK_OBJECT_CLASS`):

- `CKO_PUBLIC_KEY` — открытый асимметричный ключ;
- `CKO_PRIVATE_KEY` — приватный асимметричный ключ;
- `CKO_SECRET_KEY` — секретный симметричный ключ;
- `CKO_CERTIFICATE` — сертификат;
- `CKO_DATA` — данные.

Механизмы

Многие операции как с внутренними объектами токена, так и с внешними данными, осуществляются с помощью механизмов:

```
typedef struct CK_MECHANISM {
    CK_MECHANISM_TYPE mechanism;
    CK_VOID_PTR pParameter;
    CK_ULONG ulParameterLen;
} CK_MECHANISM;
```

где:

- `mechanism` — тип механизма (`typedef CK_ULONG CK_MECHANISM_TYPE`);
- `pParameter` — указатель на параметр механизма;
- `ulParameterLen` — размер параметра.

В ViPNet PKCS#11 VT поддерживаются следующие типы механизмов:

Таблица 5. Механизмы, поддерживаемые ViPNet PKCS#11 VT

Тип механизма	Функция	Описание
`CKM_GOSTR3410_KEY_PAIR_GEN`	`C_GenerateKeyPair`	Выработка асимметричных ключей в соответствии с ГОСТ Р 34.10-2001
`CKM_GOST28147_KEY_GEN`	`C_GenerateKey`	Выработка симметричных ключей в соответствии с ГОСТ 28147-89 для шифрования
`CKM_GOSTPBE_94`	`C_GenerateKey`	Выработка секретного ключа из пароля в соответствии с описанием генерации ключа для схемы PBES2 в документе ТК26 «Парольная защита с использованием алгоритмов ГОСТ (дополнения к PKCS#5)»
`CKM_PKCS5_PBKD`	`C_GenerateKey`	Выработка симметричного ключа в соответствии с алгоритмом PBKDF2
`CKM_GOSTR3410_512_KEY_PAIR_GEN`	`C_GenerateKeyPair`	Выработка ключей по ГОСТ Р 34.10-2012
`CKM_GOST28147_KEY_WRAP`	`C_Wrap` `C_Unwrap`	Шифрование симметричных ключей ГОСТ 28147-89 с использованием симметричных ключей ГОСТ 28147-89
`CKM_GOSTR3410_KEY_WRAP`	`C_Wrap` `C_Unwrap`	Шифрование симметричных ключей ГОСТ 28147-89 с использованием симметричного ключа ГОСТ 28147-89, выработанного по протоколу Диффи-Хеллмана из асимметричных ключей

Тип механизма	Функция	Описание
		ГОСТ Р 34.10-2001
CKM_GOSTR3410_DERIVE	C_DeriveKey	Выработка симметричных ключей ГОСТ 28147-89 с использованием асимметричных ключей ГОСТ Р 34.10-2001
CKM_GOSTR3410_PUBLIC_KEY_DERIVE	C_DeriveKey	Выработка открытого ключа ГОСТ Р 34.10-2001 по закрытому ключу
CKM_GOSTR3410_12_DERIVE	C_DeriveKey	Выработка симметричных ключей ГОСТ 28147-89 с использованием асимметричных ключей ГОСТ Р 34.10-2012
CKM_GOSTR3410_512_PUBLIC_KEY DERIVE	C_DeriveKey	Выработка открытого ключа ГОСТ Р 34.10-2012 по закрытому ключу
CKM_GOSTR3410_512_DERIVE	C_DeriveKey	Алгоритм Диффи — Хеллмана по ГОСТ Р 34.10-2012 для ключей 512 бит
CKM_KDF_4357	C_DeriveKey	Выработка диверсифицированного в соответствии с RFC 4357 симметричного ключа ГОСТ 28147-89
CKM_GOST28147_ECB	C_EncryptInit C_DecryptInit	Шифрование в соответствии с ГОСТ 28147 в режиме ECB
CKM_GOST28147	C_EncryptInit C_DecryptInit	Шифрование в соответствии с ГОСТ 28147 в режиме, отличном от ECB
CKM_GOSTR3411	C_DigestInit	Вычисление хэш-функции в соответствии с ГОСТ Р 31.11-94
CKM_GOSTR3411_12_256	C_DigestInit	Хэширование по ГОСТ Р 34.11-2012 с длиной хэша 256 бит
CKM_GOSTR3411_12_512	C_DigestInit	Хэширование по ГОСТ Р 34.11-2012 с длиной хэша 512 бит
CKM_GOST28147_MAC	C_SignInit C_VerifyInit	Вычисление MAC с использованием симметричного ключа ГОСТ 28147-89
CKM_GOSTR3411_HMAC	C_SignInit C_VerifyInit	Вычисление HMAC с использованием симметричного ключа ГОСТ 28147-89
CKM_GOSTR3411_12_256_HMAC	C_SignInit C_VerifyInit	Вычисление HMAC по ГОСТ Р 34.11-2012 со значениями B = 64, L = 32
CKM_GOSTR3411_12_512_HMAC	C_SignInit C_VerifyInit	Вычисление HMAC по ГОСТ Р 34.11-2012 со значениями B = 32, L = 64

Тип механизма	Функция	Описание
CKM_GOSTR3410	C_SignInit C_VerifyInit	Вычисление и проверка подписи с использованием асимметричного ключа ГОСТ Р 34.10-2001
CKM_GOSTR3410_WITH_GOSTR3411	C_SignInit C_VerifyInit	Вычисление и проверка подписи с использованием асимметричного ключа ГОСТ Р 34.10-2001 и с предварительной выработкой значения хэш-функции по ГОСТ Р 34.11 94
CKM_GOSTR3410_WITH_GOSTR3411_12_256	C_SignInit C_VerifyInit	Вычисление и проверка подписи с использованием асимметричного ключа ГОСТ Р 34.10-2001 и с предварительной выработкой значения хэш-функции по ГОСТ Р 34.11 2012 256 бит
CKM_GOSTR3410_512	C_SignInit C_VerifyInit	Вычисление и проверка подписи по ГОСТ Р 34.10-2012 с ключом 512 бит
CKM_GOSTR3410_WITH_GOSTR3411_12_512	C_SignInit C_VerifyInit	Вычисление и проверка подписи с использованием асимметричного ключа ГОСТ Р 34.10-2012 и с предварительной выработкой значения хэш-функции по ГОСТ Р 34.11 2012 512 бит

Сессии

Сессия — это сеанс связи с токеном пользователя. Сессии могут быть только для чтения (R/O) и для чтения/записи (R/W). После открытия сессии пользователь получает доступ к public-объектам токена. После авторизации (login) пользователь получает доступ к private-объектам токена.

В ViPNet PKCS#11 VT объекты токена (объекты постоянного хранения) хранятся в основном хранилище, в качестве которого используется файловая СУБД SQLite. Перед сохранением объекта секретные атрибуты зашифровываются и вычисляется его имитозащитная вставка, при извлечении объекта секретные атрибуты расшифровываются и проверяется имитозащитная вставка.

Временные объекты хранятся в кэше. Кэш объектов — это дополнительный слой над хранилищем объектов, необходимый для хранения временных (сессионных) объектов, кэширования приватных объектов (уже расшифрованных), кэширования общедоступных объектов. Кэш является общим для всех объектов токена. В настоящее время используется простейшая политика кэширования — при получении объекта из базы данных он сохраняется для дальнейшего использования в кэше. Объекты хранятся в кэше до выполнения процедуры `logout` или закрытия сессии. Кэш может быть отключен в файле настроек модуля `pkcs11`.

В соответствии с общим стандартом PKCS#11 v2.20, ViPNet PKCS#11 VT поддерживает режимы нескольких сессий с несколькими токенами (multiple sessions on multiple tokens). При этом каждая сессия может находиться в нескольких состояниях. Состояние сессии определяет возможности доступа к объектам и те функции, которые могут быть выполнены с этими объектами.

Таблица 6. Состояние R/O сессий

Состояние	Описание
R/O Public Session	Приложение открыло R/O сессию. Приложение получает R/O доступ к public-объектам токена и R/W доступ к объектам public session.
R/O User Functions	Обычный пользователь (не администратор) авторизовался, то есть осуществил login. Приложение получает R/O доступ ко всем объектам токена (public or private) и R/W доступ ко всем session объектам (public or private).

Таблица 7. Состояние R/W сессий

Состояние	Описание
R/W Public Session	Приложение открыло R/W сессию. Приложение получает R/W доступ ко всем public-объектам.
R/W SO Functions	Администратор авторизовался и подключился к токену. Приложение получает R/W доступ только к public объектам токена, но не к private объектам. Администратор может установить ПИН-код обычного пользователя.
R/W User Functions	Обычный пользователь авторизовался, то есть осуществил login. Приложение получает R/W доступ ко всем объектам.

Приведенная ниже таблица суммирует виды доступа к различным объектам при различных состояниях сессии.

Таблица 8. Доступ к различным типам объектов для различных типов сессий

	Тип сессии				
Type of object	R/O Public	R/W Public	R/O User	R/W User	R/W SO
Public session object	R/W	R/W	R/W	R/W	R/W
Private session object			R/W	R/W	
Public token object	R/O	R/W	R/O	R/W	R/W
Private token object			R/O	R/W	

3

Особенности использования интерфейса PKCS#11

Общие особенности

При выполнении `C_Initialize` ViPNet PKCS#11 VT создает дополнительный рабочий поток. Без этого потока ViPNet PKCS#11 VT корректно работать не может, поэтому интерфейс не может быть использован в условиях, когда невозможно или нельзя создавать потоки.

Внимание! Если был выполнен `C_Initialize`, нельзя использовать `fork()` для порождения нового процесса, использующего ViPNet PKCS#11 VT. `fork()` процесса можно делать только до вызова `C_Initialize`.

Если при записи или изменении объектов на токене не хватает скорости (скорость записи на жестком диске — порядка 10 операций в секунду), то запись следует вести в многопоточном режиме. В этом случае скорость записи будет расти линейно пропорционально количеству потоков. Особенности многопоточной записи см. в разделе Особенности многопоточного использования (на стр. 28).

Особенности использования функций поиска

Если вы работаете с токенами, на которых хранится более 1000 объектов, при поиске объектов на токене (с помощью `C_FindObjects`, `CKA_TOKEN == CK_TRUE`) во избежание возникновения значительных потерь в скорости поиска необходимо учитывать следующие особенности:

1. Шаблон поиска должен содержать минимально необходимый набор атрибутов для нахождения необходимого объекта, так как:
   - Чем больше атрибутов в шаблоне, тем поиск медленнее.
   - Чем больше найденных объектов по заданному шаблону, тем поиск медленнее.

   Пример (идеальный вариант)

```
CK_BBOOL ckTrue = CK_TRUE;
CK_ULONG id = 12345;
CK_ATTRIBUTE searchTemplate[] =
{
{ CKA_TOKEN, &ckTrue, sizeof(ckTrue) },
{ CKA_ID, &id, sizeof(id) }
};
CK_ULONG ulObjectCount;
CK_RV rv;
rv = C_FindObjectsInit(hSession, NULL_PTR, 0);
assert(rv == CKR_OK);
```

2. Шаблон поиска должен содержать один из индексирующих атрибутов:
   - `CKA_ID`
   - `CKA_LABEL`
   - `CKA_SUBJECT`
   - `CKA_ISSUER`
   - `CKA_VALUE`

   Желательно, чтобы по этому атрибуту можно было однозначно определить искомый объект (или хотя бы не более 10 объектов). Например, OpenSSL использует `CKA_ID` в качестве уникального идентификатора ключевой пары (а также для связи с ней сертификата).

Особенности многопоточного использования

Интерфейс ViPNet PKCS#11 VT обеспечивает корректную многопоточную работу при условии установки корректных аргументов в `C_Initialize` (на стр. 33).

При работе с объектами на токене (`CKA_TOKEN == CK_TRUE`) в многопоточном режиме следует учитывать следующую особенность: если одновременно производить операции записи из нескольких потоков (например, `C_CreateObject` или `C_SetAttributeValue`), то при возникновении ошибки записи на токен (именно при записи в файл) в одной операции, операции записи в других потоках могут также завершиться с ошибками.

Особенности многопроцессного использования

Существует несколько особенностей многопроцессного использования ViPNet PKCS#11 VT:

1. Синхронизация объектов из разных процессов: если объект создан или изменен в одном процессе (процесс 1), то в другом процессе (процесс 2) эти изменения могут быть обнаружены не сразу, если в (процессе 2) уже был вызван `C_Initialize`. Например, если изменить значение атрибута в объекте токена в процессе 1, то в процессе 2 будет по-прежнему использоваться старое значение атрибута.
2. Смена любого ПИН-кода в одном из процессов приведет к неопределенному поведению в других процессах, работающих с этим токеном, если в них уже был вызван `C_Initialize`.

4

Функции

Функции подключения и инициализации интерфейса ViPNet PKCS#11 VT

Список функций подключения и инициализации интерфейса ViPNet PKCS#11 VT:

- `C_GetFunctionList` (на стр. 31);
- `C_Initialize` (см. «C_Initialize» на стр. 33);
- `C_Finalize` (на стр. 34);
- `C_GetInfo` (см. «C_GetInfo» на стр. 34).

Функции подключения и инициализации интерфейса ViPNet PKCS#11 VT являются начальными операциями, которые должна выполнить прикладная программа для использования этого интерфейса. Они, как правило, выполняются только один раз в начале (`C_GetFunctionList`, `C_Initialize`) и в конце (`C_Finalize`) использования интерфейса.

C_GetFunctionList

```
CK_DEFINE_FUNCTION(CK_RV, C_GetFunctionList)(
    CK_FUNCTION_LIST_PTR_PTR ppFunctionList
);
```

Возвращаемым параметром функции `C_GetFunctionList` является указатель списка адресов функций интерфейса ViPNet PKCS#11 VT. Использование этой функции значительно облегчает программистам получение необходимых адресов функций из библиотеки `softtoken_pkcs11`.

Параметр `ppFunctionList` является указателем на указатель структуры `CK_FUNCTION_LIST` вида:

```
typedef struct CK_FUNCTION_LIST {
    CK_VERSION version;
    CK_C_Initialize C_Initialize;
    CK_C_Finalize C_Finalize;
    CK_C_GetInfo C_GetInfo;
    CK_C_GetFunctionList C_GetFunctionList;
    CK_C_GetSlotList C_GetSlotList;
    CK_C_GetSlotInfo C_GetSlotInfo;
    CK_C_GetTokenInfo C_GetTokenInfo;
    CK_C_GetMechanismList C_GetMechanismList;
    CK_C_GetMechanismInfo C_GetMechanismInfo;
    CK_C_InitToken C_InitToken;
    CK_C_InitPIN C_InitPIN;
    CK_C_SetPIN C_SetPIN;
    CK_C_OpenSession C_OpenSession;
```

```
CK_C_CloseSession C_CloseSession;
CK_C_CloseAllSessions C_CloseAllSessions;
CK_C_GetSessionInfo C_GetSessionInfo;
CK_C_GetOperationState C_GetOperationState;
CK_C_SetOperationState C_SetOperationState;
CK_C_Login C_Login;
CK_C_Logout C_Logout;
CK_C_CreateObject C_CreateObject;
CK_C_CopyObject C_CopyObject;
CK_C_DestroyObject C_DestroyObject;
CK_C_GetObjectSize C_GetObjectSize;
CK_C_GetAttributeValue C_GetAttributeValue;
CK_C_SetAttributeValue C_SetAttributeValue;
CK_C_FindObjectsInit C_FindObjectsInit;
CK_C_FindObjects C_FindObjects;
CK_C_FindObjectsFinal C_FindObjectsFinal;
CK_C_EncryptInit C_EncryptInit;
CK_C_Encrypt C_Encrypt;
CK_C_EncryptUpdate C_EncryptUpdate;
CK_C_EncryptFinal C_EncryptFinal;
CK_C_DecryptInit C_DecryptInit;
CK_C_Decrypt C_Decrypt;
CK_C_DecryptUpdate C_DecryptUpdate;
CK_C_DecryptFinal C_DecryptFinal;
CK_C_DigestInit C_DigestInit;
CK_C_Digest C_Digest;
CK_C_DigestUpdate C_DigestUpdate;
CK_C_DigestKey C_DigestKey;
CK_C_DigestFinal C_DigestFinal;
CK_C_SignInit C_SignInit;
CK_C_Sign C_Sign;
CK_C_SignUpdate C_SignUpdate;
CK_C_SignFinal C_SignFinal;
CK_C_SignRecoverInit C_SignRecoverInit;
CK_C_SignRecover C_SignRecover;
CK_C_VerifyInit C_VerifyInit;
CK_C_Verify C_Verify;
CK_C_VerifyUpdate C_VerifyUpdate;
CK_C_VerifyFinal C_VerifyFinal;
CK_C_VerifyRecoverInit C_VerifyRecoverInit;
CK_C_VerifyRecover C_VerifyRecover;
CK_C_DigestEncryptUpdate C_DigestEncryptUpdate;
CK_C_DecryptDigestUpdate C_DecryptDigestUpdate;
CK_C_SignEncryptUpdate C_SignEncryptUpdate;
CK_C_DecryptVerifyUpdate C_DecryptVerifyUpdate;
CK_C_GenerateKey C_GenerateKey;
CK_C_GenerateKeyPair C_GenerateKeyPair;
CK_C_WrapKey C_WrapKey;
CK_C_UnwrapKey C_UnwrapKey;
CK_C_DeriveKey C_DeriveKey;
CK_C_SeedRandom C_SeedRandom;
CK_C_GenerateRandom C_GenerateRandom;
```

```
    CK_C_GetFunctionStatus C_GetFunctionStatus;
    CK_C_CancelFunction C_CancelFunction;
    CK_C_WaitForSlotEvent C_WaitForSlotEvent;
} CK_FUNCTION_LIST;
```

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Пример:

```
CK_FUNCTION_LIST_PTR pFunctionList;
CK_C_Initialize pC_Initialize;
CK_RV rv;
/* It's OK to call C_GetFunctionList before calling C_Initialize */
rv = C_GetFunctionList(&pFunctionList);
assert(rv == CKR_OK);
pC_Initialize = pFunctionList -> C_Initialize;
/* Call the C_Initialize function in the library */
rv = (*pC_Initialize)(NULL_PTR);
```

C_Initialize

```
CK_DEFINE_FUNCTION(CK_RV, C_Initialize)(
    CK_VOID_PTR pInitArgs
);
```

`C_Initialize` инициализирует библиотеку `softtoken_pkcs11`. Параметр `pInitArgs` должен быть заполнен в соответствии со стандартом PKCS#11:

1. Если `pInitArgs` равен `NULL`, то в библиотеке отключаются блокировки и она не может быть использована в многопоточном режиме.
2. Если `pInitArgs` указывает на структуру `CK_C_INITIALIZE_ARGS`:
   - **2.1** Поле `pReserved` должно быть равно `NULL`.
   - **2.2** Если `flags` содержит `CKF_OS_LOCKING_OK` или заданы все функции `CreateMutex`, `DestroyMutex`, `LockMutex`, `UnlockMutex`, то в библиотеке включаются блокировки и она может быть использована в многопоточном режиме.
   - **2.3** Если `flags` не содержит `CKF_OS_LOCKING_OK` и не заданы все функции блокировок, то в библиотеке отключаются блокировки и она не может быть использована в многопоточном режиме.
   - **2.4** Если задан флаг `CKF_LIBRARY_CANT_CREATE_OS_THREADS`, то библиотека возвращает ошибку, так как для корректного функционирования ей необходимо создавать потоки (см. «Особенности использования интерфейса PKCS#11» на стр. 25).

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CANT_LOCK`, `CKR_CRYPTOKI_ALREADY_INITIALIZED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_NEED_TO_CREATE_THREADS`, `CKR_OK`.

Пример: см. раздел C_GetInfo (на стр. 34).

C_Finalize

```
CK_DEFINE_FUNCTION(CK_RV, C_Finalize)(
    CK_VOID_PTR pReserved
);
```

Функция `C_Finalize` вызывается для завершения работы с интерфейсом ViPNet PKCS#11 VT и освобождения используемых им ресурсов. Параметр `pReserved` зарезервирован и должен быть равен `NULL_PTR`.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

C_GetInfo

```
CK_DEFINE_FUNCTION(CK_RV, C_GetInfo)(
    CK_INFO_PTR pInfo
);
```

`C_GetInfo` вызывается для получения общей информации об интерфейсе ViPNet PKCS#11 VT. В качестве параметра используется стандартная структура `CK_INFO`:

```
typedef struct CK_INFO {
    /* manufacturerID and libraryDecription have been changed from
      * CK_CHAR to CK_UTF8CHAR for v2.10 */
    CK_VERSION    cryptokiVersion;   /* Cryptoki interface ver */
    CK_UTF8CHAR   manufacturerID[32]; /* blank padded */
    CK_FLAGS      flags;        /* must be zero */
    /* libraryDescription and libraryVersion are new for v2.0 */
    CK_UTF8CHAR   libraryDescription[32]; /* blank padded */
    CK_VERSION    libraryVersion;     /* version of library */
} CK_INFO;
```

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Пример исходного кода:

```
CK_INFO info;
CK_RV rv;
CK_C_INITIALIZE_ARGS InitArgs;
InitArgs.CreateMutex = &MyCreateMutex;
InitArgs.DestroyMutex = &MyDestroyMutex;
InitArgs.LockMutex = &MyLockMutex;
InitArgs.UnlockMutex = &MyUnlockMutex;
InitArgs.flags = CKF_OS_LOCKING_OK;
InitArgs.pReserved = NULL_PTR;
rv = C_Initialize((CK_VOID_PTR)&InitArgs);
assert(rv == CKR_OK);
rv = C_GetInfo(&info);
assert(rv == CKR_OK);
```

```
if(info.cryptokiVersion.major == 2) {
    /* Do lots of interesting cryptographic things with the token */
.
.
}
rv = C_Finalize(NULL_PTR);
assert(rv == CKR_OK);
```

Функции получения информации о слотах, токенах и механизмах

Список функций получения информации о слотах, токенах и механизмах:

- `C_GetSlotList` (на стр. 36);
- `C_GetSlotInfo` (на стр. 37);
- `C_GetTokenInfo` (на стр. 38);
- `C_WaitForSlotEvent` (на стр. 39);
- `C_GetMechanismList` (на стр. 39);
- `C_GetMechanismInfo` (на стр. 40).

Функции получения информации о слотах, токенах и механизмах позволяют прикладной программе получать различную информацию о слотах и токенах, а также о поддерживаемых ViPNet PKCS#11 VT механизмах. Как правило, эти функции вызываются до открытия сессии, с тем чтобы выбрать токен, для которого необходимо открыть сессию, или проверить, что токен поддерживает требуемый прикладной программе механизм.

C_GetSlotList

```
CK_DEFINE_FUNCTION(CK_RV, C_GetSlotList)(
    CK_BBOOL tokenPresent,
    CK_SLOT_ID_PTR pSlotList,
    CK_ULONG_PTR pulCount
);
```

Функция `C_GetSlotList` вызывается для получения списка всех доступных слотов. Параметр `tokenPresent` позволяет получать либо список всех слотов (`tokenPresent` = `CK_FALSE`), либо только слотов, содержащих токен (`tokenPresent` = `CK_TRUE`).

Результат — указатель `pSlotList` — указывает на перечень идентификаторов доступных слотов (`typedef CK_ULONG CK_SLOT_ID`).

Функция `C_GetSlotList` удовлетворяет общепринятому соглашению об использовании массивов неопределенной длины, а именно:

- если параметр `pSlotList` равен `NULL_PTR`, то указатель `pulCount` возвращает требуемый размер массива `pSlotList`;
- если параметр `pSlotList` не равен `NULL_PTR`, а указатель `pulCount` указывает на размер, меньший требуемого, то функция ничего не записывает в `pSlotList`, а возвращает требуемый размер в `pulCount` и код `CKR_BUFFER_TOO_SMALL`;

- если параметр `pSlotList` не равен `NULL_PTR`, а указатель `pulCount` указывает на размер, больший либо равный требуемому, функция записывает перечень идентификаторов в `pSlotList` и возвращает истинный размер списка в `pulCount` и код `CKR_OK`.

Аналогичное соглашение будет часто использоваться в других подобных функциях, поэтому в дальнейшем оно будет именоваться как «соглашение МНД», то есть соглашение об использовании массивов неопределенной длины.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Пример:

```
CK_ULONG ulSlotCount, ulSlotWithTokenCount;
CK_SLOT_ID_PTR pSlotList, pSlotWithTokenList;
CK_RV rv;
/* Get list of all slots */
rv = C_GetSlotList(CK_FALSE, NULL_PTR, &ulSlotCount);
if (rv == CKR_OK) {
    pSlotList =
        (CK_SLOT_ID_PTR) malloc(ulSlotCount*sizeof(CK_SLOT_ID));
    rv = C_GetSlotList(CK_FALSE, pSlotList, &ulSlotCount);
    if (rv == CKR_OK) {
        /* Now use that list of all slots */
        .
        .
    }
    free(pSlotList);
}
/* Get list of all slots with a token present */
pSlotWithTokenList = (CK_SLOT_ID_PTR) malloc(0);
ulSlotWithTokenCount = 0;
while (1) {
    rv = C_GetSlotList(
        CK_TRUE, pSlotWithTokenList, ulSlotWithTokenCount);
    if (rv != CKR_BUFFER_TOO_SMALL)
        break;
    pSlotWithTokenList = realloc(
        pSlotWithTokenList,
        ulSlotWithTokenList*sizeof(CK_SLOT_ID));
}
if (rv == CKR_OK) {
    /* Now use that list of all slots with a token present */
    .
    .
}
free(pSlotWithTokenList);
```

C_GetSlotInfo

```
CK_DEFINE_FUNCTION(CK_RV, C_GetSlotInfo)(
```

```
    CK_SLOT_ID slotID,
    CK_SLOT_INFO_PTR pInfo
);
```

Функция `C_GetSlotInfo` позволяет приложению получать информацию об определенном слоте. Информация представляется в виде структуры `CK_SLOT_INFO`.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK` и `CKR_SLOT_ID_INVALID`.

Пример: см. раздел C_GetTokenInfo (на стр. 38).

C_GetTokenInfo

```
CK_DEFINE_FUNCTION(CK_RV, C_GetTokenInfo)(
    CK_SLOT_ID slotID,
    CK_TOKEN_INFO_PTR pInfo
);
```

Функция `C_GetTokenInfo` позволяет приложению получать информацию о подключенном к слоту токене. Информация представляется в виде структуры `CK_TOKEN_INFO`.

Параметры:

- `slotID` — идентификатор слота с токеном;
- `pInfo` — указатель структуру `CK_TOKEN_INFO`, в которую будет помещен результат.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_SLOT_ID_INVALID`, `CKR_TOKEN_NOT_PRESENT`, `CKR_TOKEN_NOT_RECOGNIZED`, `CKR_ARGUMENTS_BAD`.

Пример:

```
CK_ULONG ulCount;
CK_SLOT_ID_PTR pSlotList;
CK_SLOT_INFO slotInfo;
CK_TOKEN_INFO tokenInfo;
CK_RV rv;
rv = C_GetSlotList(CK_FALSE, NULL_PTR, &ulCount);
if ((rv == CKR_OK) && (ulCount > 0)) {
    pSlotList = (CK_SLOT_ID_PTR) malloc(ulCount*sizeof(CK_SLOT_ID));
    rv = C_GetSlotList(CK_FALSE, pSlotList, &ulCount);
    assert(rv == CKR_OK);
    /* Get slot information for first slot */
    rv = C_GetSlotInfo(pSlotList[0], &slotInfo);
    assert(rv == CKR_OK);
    /* Get token information for first slot */
    rv = C_GetTokenInfo(pSlotList[0], &tokenInfo);
    if (rv == CKR_TOKEN_NOT_PRESENT) {
        .
        .
```

```
    }
    .
    .
    free(pSlotList);
}
```

C_WaitForSlotEvent

```
CK_DEFINE_FUNCTION(CK_RV, C_WaitForSlotEvent)(
    CK_FLAGS flags,
    CK_SLOT_ID_PTR pSlot,
    CK_VOID_PTR pReserved
);
```

Поскольку токены в данной реализации не могут удаляться из слотов, то события, связанные с PKCS11, не поддерживаются. Функция все время возвращает `CKR_NO_EVENT`.

Код возврата: `CKR_NO_EVENT`.

Пример:

```
CK_FLAGS flags = 0;
CK_SLOT_ID slotID;
CK_SLOT_INFO slotInfo;
.
.
/* Block and wait for a slot event */
rv = C_WaitForSlotEvent(flags, &slotID, NULL_PTR);
assert(rv == CKR_OK);
/* See what's up with that slot */
rv = C_GetSlotInfo(slotID, &slotInfo);
assert(rv == CKR_OK);
```

C_GetMechanismList

```
CK_DEFINE_FUNCTION(CK_RV, C_GetMechanismList)(
    CK_SLOT_ID slotID,
    CK_MECHANISM_TYPE_PTR pMechanismList,
    CK_ULONG_PTR pulCount
);
```

Функция `C_GetMechanismList` позволяет прикладной программе получать перечень всех механизмов, поддерживаемых токеном. В ViPNet PKCS#11 VT этот перечень одинаков для всех токенов. Функция поддерживает соглашение МНД.

Параметры:

- `slotID` — идентификатор слота с токеном;
- `pMechanismList` — указатель, по которому выдается список механизмов;

- `pulCount` — указатель на размер списка.

Коды возврата: `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_SLOT_ID_INVALID`, `CKR_TOKEN_NOT_PRESENT`, `CKR_TOKEN_NOT_RECOGNIZED`, `CKR_ARGUMENTS_BAD`.

Пример:

```
CK_SLOT_ID slotID;
CK_ULONG ulCount;
CK_MECHANISM_TYPE_PTR pMechanismList;
CK_RV rv;
.
.
rv = C_GetMechanismList(slotID, NULL_PTR, &ulCount);
if ((rv == CKR_OK) && (ulCount > 0)) {
    pMechanismList =
         (CK_MECHANISM_TYPE_PTR)
         malloc(ulCount*sizeof(CK_MECHANISM_TYPE));
    rv = C_GetMechanismList(slotID, pMechanismList, &ulCount);
    if (rv == CKR_OK) {
         .
         .
    }
    free(pMechanismList);
}
```

C_GetMechanismInfo

```
CK_DEFINE_FUNCTION(CK_RV, C_GetMechanismInfo)(
    CK_SLOT_ID slotID,
    CK_MECHANISM_TYPE type,
    CK_MECHANISM_INFO_PTR pInfo
);
```

`C_GetMechanismInfo` позволяет получать расширенную информацию о механизме, возвращаемую в виде структуры `CK_MECHANISM_INFO`:

```
typedef struct CK_MECHANISM_INFO {
        CK_ULONG    ulMinKeySize;
        CK_ULONG    ulMaxKeySize;
        CK_FLAGS    flags;
} CK_MECHANISM_INFO;
/* The flags are defined as follows:
  *      Bit Flag              Mask        Meaning */
#define CKF_HW                0x00000001 /* performed by HW */
/* The flags CKF_ENCRYPT, CKF_DECRYPT, CKF_DIGEST, CKF_SIGN,
  * CKG_SIGN_RECOVER, CKF_VERIFY, CKF_VERIFY_RECOVER,
  * CKF_GENERATE, CKF_GENERATE_KEY_PAIR, CKF_WRAP, CKF_UNWRAP,
  * and CKF_DERIVE are new for v2.0.  They specify whether or not
  * a mechanism can be used for a particular task */
```

```
#define CKF_ENCRYPT            0x00000100
#define CKF_DECRYPT            0x00000200
#define CKF_DIGEST             0x00000400
#define CKF_SIGN               0x00000800
#define CKF_SIGN_RECOVER       0x00001000
#define CKF_VERIFY             0x00002000
#define CKF_VERIFY_RECOVER     0x00004000
#define CKF_GENERATE           0x00008000
#define CKF_GENERATE_KEY_PAIR  0x00010000
#define CKF_WRAP               0x00020000
#define CKF_UNWRAP             0x00040000
#define CKF_DERIVE             0x00080000
/* CKF_EC_F_P, CKF_EC_F_2M, CKF_EC_ECPARAMETERS, CKF_EC_NAMEDCURVE,
  * CKF_EC_UNCOMPRESS, and CKF_EC_COMPRESS are new for v2.11. They
  * describe a token's EC capabilities not available in mechanism
  * information. */
#define CKF_EC_F_P             0x00100000
#define CKF_EC_F_2M            0x00200000
#define CKF_EC_ECPARAMETERS    0x00400000
#define CKF_EC_NAMEDCURVE      0x00800000
#define CKF_EC_UNCOMPRESS      0x01000000
#define CKF_EC_COMPRESS        0x02000000
#define CKF_EXTENSION          0x80000000 /* FALSE for this version */
```

Параметры:

- `slotID` — идентификатор слота с токеном;
- `type` — тип механизма;
- `pInfo` — указатель на структуру с информацией о механизме.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_MECHANISM_INVALID`, `CKR_OK`, `CKR_SLOT_ID_INVALID`, `CKR_TOKEN_NOT_PRESENT`, `CKR_TOKEN_NOT_RECOGNIZED`, `CKR_ARGUMENTS_BAD`.

Пример:

```
CK_SLOT_ID slotID;
CK_MECHANISM_INFO info;
CK_RV rv;
.
.
/* Get information about the CKM_GOSTR3411 mechanism for this token */
rv = C_GetMechanismInfo(slotID, CKM_GOSTR3411, &info);
if (rv == CKR_OK) {
    if (info.flags & CKF_DIGEST) {
        .
        .
    }
}
```

Функции администрирования

Список функций администрирования:

- `C_InitToken` (на стр. 42);
- `C_InitPIN` (на стр. 43);
- `C_SetPIN` (на стр. 43).

Функции администрирования позволяют инициализировать токен, а также устанавливать или изменять ПИН-код администратора или пользователя.

C_InitToken

```
CK_DEFINE_FUNCTION(CK_RV, C_InitToken)(
    CK_SLOT_ID slotID,
    CK_UTF8CHAR_PTR pPin,
    CK_ULONG ulPinLen,
    CK_UTF8CHAR_PTR pLabel
);
```

В ViPNet PKCS#11 VT функция `C_InitToken` осуществляет следующие действия:

- удаляет всю информацию на токене (если она была);
- добавляет к информации токена метку `pLabel`;
- добавляет к флагам токена `CKF_TOKEN_INITIALIZED`.

Параметрами функции `C_InitToken` являются ID слота с токеном, ПИН-код администратора и его длина, а также метка токена.

Примечание. В ViPNet PKCS#11 VT токен создается с инициализированным ПИН-кодом администратора, равным `11111111`. Рекомендуется сменить ПИН-код на более надежный.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_PIN_INCORRECT`, `CKR_PIN_LOCKED`, `CKR_SESSION_EXISTS`, `CKR_SLOT_ID_INVALID`, `CKR_TOKEN_NOT_PRESENT`, `CKR_TOKEN_NOT_RECOGNIZED`, `CKR_TOKEN_WRITE_PROTECTED`, `CKR_ARGUMENTS_BAD`.

Пример:

```
CK_SLOT_ID slotID;
CK_UTF8CHAR_PTR pin = "MyPIN";
CK_UTF8CHAR label[32];
CK_RV rv;
```

```
.
.
memset(label, ' ', sizeof(label));
memcpy(label, "My first token", strlen("My first token"));
rv = C_InitToken(slotID, pin, strlen(pin), label);
if (rv == CKR_OK) {
    .
    .
}
```

C_InitPIN

```
CK_DEFINE_FUNCTION(CK_RV, C_InitPIN)(
    CK_SESSION_HANDLE hSession,
    CK_UTF8CHAR_PTR pPin,
    CK_ULONG ulPinLen
);
```

Функция `C_InitPIN` позволяет администратору установить ПИН-код пользователя, причем это может быть как установка ПИН-кода для нового токена, не содержащего никакой приватной информации пользователя, так и установка нового ПИН-кода пользователя в случае, если он забыл старый ПИН-код. В последнем случае сохраняется вся открытая информация токена, но вся приватная информация будет удалена и безвозвратно утеряна.

Параметрами функции являются указатель сессии и ПИН-код администратора. Сессия должна находиться в состоянии `R/W SO Functions`.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_PIN_INVALID`, `CKR_PIN_LEN_RANGE`, `CKR_SESSION_CLOSED`, `CKR_SESSION_READ_ONLY`, `CKR_SESSION_HANDLE_INVALID`, `CKR_TOKEN_WRITE_PROTECTED`, `CKR_USER_NOT_LOGGED_IN`, `CKR_ARGUMENTS_BAD`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_UTF8CHAR newPin[]= {"NewPIN"};
CK_RV rv;
rv = C_InitPIN(hSession, newPin, sizeof(newPin));
if (rv == CKR_OK) {
    .
    .
}
```

C_SetPIN

```
CK_DEFINE_FUNCTION(CK_RV, C_SetPIN)(
    CK_SESSION_HANDLE hSession,
    CK_UTF8CHAR_PTR pOldPin,
```

```
    CK_ULONG ulOldLen,
    CK_UTF8CHAR_PTR pNewPin,
    CK_ULONG ulNewLen
);
```

Функция `C_SetPIN` позволяет изменять ПИН-код администратора или пользователя. Смена ПИН-кода зависит от состояния сессии.

Таблица 9. Зависимость ПИН-кода от состояния сессии

Состояние сессии	Изменяемый ПИН-код
`R/W Public Session`	ПИН-код пользователя
`R/W User Functions`	ПИН-код пользователя
`R/W SO Functions`	ПИН-код администратора

Параметрами являются идентификатор сессии, а также старый и новый ПИН-коды и их длины.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_PIN_INCORRECT`, `CKR_PIN_INVALID`, `CKR_PIN_LEN_RANGE`, `CKR_PIN_LOCKED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SESSION_READ_ONLY`, `CKR_TOKEN_WRITE_PROTECTED`, `CKR_ARGUMENTS_BAD`.

Функции открытия и закрытия сессий и их авторизации

Список функций открытия и закрытия сессий и их авторизации:

- `C_OpenSession` (на стр. 45);
- `C_CloseSession` (на стр. 46);
- `C_CloseAllSessions` (на стр. 47);
- `C_GetSessionInfo` (на стр. 47);
- `C_Login` (на стр. 48);
- `C_Logout` (на стр. 48).

Функции из этого раздела осуществляют открытие и закрытие сессий, то есть сеансов связи с различными токенами, а также авторизацию пользователя или администратора, необходимую для выполнения тех или иных операций с приватными данными токена.

C_OpenSession

```
CK_DEFINE_FUNCTION(CK_RV, C_OpenSession)(
    CK_SLOT_ID slotID,
    CK_FLAGS flags,
    CK_VOID_PTR pApplication,
    CK_NOTIFY Notify,
    CK_SESSION_HANDLE_PTR phSession
);
```

Функция `C_OpenSession` является одной из основных функций интерфейса PKCS#11, поскольку она открывает сеанс связи с токеном.

Параметры:

- перед применением этой функции прикладная программа каким-то образом должна выбрать слот, содержащий нужный токен, и получить его идентификатор — параметр `slotID`.
- `flags` — набор флагов сессии. Значения параметра приведены в таблице ниже.

Таблица 10. Значения параметра `flags`

Флаг	Маска	Значение
CKF_RW_SESSION	0x00000002	Сессия является R/W (для записи/чтения). Если флаг не установлен, то сессия является сессией только для чтения (R/O).
CKF_SERIAL_SESSION	0x00000004	Этот флаг всегда должен присутствовать. Если он отсутствует, то ViPNet PKCS#11 VT возвращает ошибку CKR_SESSION_PARALLEL_NOT_SUPPORTED.

- `pApplication` — установленный внешним приложением указатель, параметр функции обратной связи. В ViPNet PKCS#11 VT эти параметры не поддерживаются;
- `Notify` — адрес функции обратной связи. В ViPNet PKCS#11 VT эти параметры не поддерживаются;
- `phSession` — указатель на дескриптор (`handle`) сессии.

Коды возврата: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SESSION_COUNT, CKR_SESSION_PARALLEL_NOT_SUPPORTED, CKR_SESSION_READ_WRITE_SO_EXISTS, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_RECOGNIZED, CKR_TOKEN_WRITE_PROTECTED, CKR_ARGUMENTS_BAD.

Пример: см. раздел C_CloseSession (на стр. 46).

C_CloseSession

```
CK_DEFINE_FUNCTION(CK_RV, C_CloseSession)(
    CK_SESSION_HANDLE hSession
);
```

Функция `C_CloseSession` закрывает сессию с дескриптором `hSession`.

Коды возврата: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример:

```
CK_SLOT_ID slotID;
CK_BYTE application;
CK_NOTIFY MyNotify;
CK_SESSION_HANDLE hSession;
CK_RV rv;
.
.
application = 17;
```

```
MyNotify = &EncryptionSessionCallback;
rv = C_OpenSession(
    slotID, CKF_SERIAL_SESSION | CKF_RW_SESSION,
        (CK_VOID_PTR) &application, MyNotify,
    &hSession);
if (rv == CKR_OK) {
    .
    .
    C_CloseSession(hSession);
}
```

C_CloseAllSessions

```
CK_DEFINE_FUNCTION(CK_RV, C_CloseAllSessions)(
    CK_SLOT_ID slotID
);
```

Функция `C_CloseAllSessions` закрывает все сессии к слоту с идентификатором `slotID`.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_SLOT_ID_INVALID`, `CKR_TOKEN_NOT_PRESENT`.

C_GetSessionInfo

```
CK_DEFINE_FUNCTION(CK_RV, C_GetSessionInfo)(
    CK_SESSION_HANDLE hSession,
    CK_SESSION_INFO_PTR pInfo
);
```

Функция `C_GetSessionInfo` возвращает информацию о сессии с дескриптором `hSession` в структуру `CK_SESSION_INFO`:

```
/* CK_SESSION_INFO provides information about a session */
typedef struct CK_SESSION_INFO {
    CK_SLOT_ID    slotID;
    CK_STATE      state;     /* see here */
    CK_FLAGS      flags;     /* see here */
    /* ulDeviceError was changed from CK_USHORT to CK_ULONG for
      * v2.0 */
/* device-dependent error code - always 0 in ViPNet PKCS#11 VT*/
    CK_ULONG      ulDeviceError;
} CK_SESSION_INFO;
```

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_ARGUMENTS_BAD`.

C_Login

```
    CK_SESSION_HANDLE hSession,
    CK_USER_TYPE userType,
    CK_UTF8CHAR_PTR pPin,
    CK_ULONG ulPinLen
);
```

Функция `C_Login` осуществляет авторизацию пользователя или администратора в ходе сессии с дескриптором `hSession`. Параметр `userType` определяет тип пользователя. Для ViPNet PKCS#11 VT поддерживаются следующие типы пользователей:

- `CKU_SO` — администратор;
- `CKU_USER` — обычный пользователь.

Параметры `pPin` и `ulPinLen` определяют ПИН-код.

Результатом выполнения функции `C_Login` является изменение состояния всех открытых к данному токену сессий.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_PIN_INCORRECT`, `CKR_PIN_LOCKED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SESSION_READ_ONLY_EXISTS`, `CKR_USER_ALREADY_LOGGED_IN`, `CKR_USER_ANOTHER_ALREADY_LOGGED_IN`, `CKR_USER_PIN_NOT_INITIALIZED`, `CKR_USER_TOO_MANY_TYPES`, `CKR_USER_TYPE_INVALID`.

Пример: см. раздел C_Logout (на стр. 48).

C_Logout

```
CK_DEFINE_FUNCTION(CK_RV, C_Logout)(
    CK_SESSION_HANDLE hSession
);
```

Функция `C_Logout` прекращает авторизованный доступ к приватным объектам токена, но не закрывает сессии. Результатом выполнения функции `C_Logout` является изменение состояния всех открытых к данному токену сессий.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_UTF8CHAR userPIN[] = {“MyPIN”};
CK_RV rv;
```

```
rv = C_Login(hSession, CKU_USER, userPIN, sizeof(userPIN));
if (rv == CKR_OK) {
    .
    .
    rv == C_Logout(hSession);
    if (rv == CKR_OK) {
        .
        .
    }
}
```

Функции сохранения и восстановления состояния сессии

Общие сведения

Функции из этого раздела осуществляют сохранение и восстановление состояния сессии, в которой запущена криптографическая операция.

Список операций и механизмов, состояние которых может быть сохранено и восстановлено:

- Операции типа `C_Sign` (сессия, в которой был вызов `C_SignInit`): `CKM_GOSTR3411_HMAC`, `CKM_GOSTR3411_12_256_HMAC`, `CKM_GOSTR3411_12_512_HMAC`, `CKM_GOST28147_MAC`.
- Операции типа `C_Verify` (сессия, в которой был вызов `C_VerifyInit`): `CKM_GOSTR3411_HMAC`, `CKM_GOSTR3411_12_256_HMAC`, `CKM_GOSTR3411_12_512_HMAC`, `CKM_GOST28147_MAC`.
- Операции типа `C_Digest` (сессия, в которой был вызов `C_DigestInit`): `CKM_GOSTR3411`, `CKM_GOSTR3411_12_256`, `CKM_GOSTR3411_12_512`.

C_GetOperationState

```
CK_DEFINE_FUNCTION(CK_RV, C_GetOperationState)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pOperationState,
    CK_ULONG_PTR pulOperationStateLen
);
```

Функция `C_GetOperationState` сохраняет состояние сессии `hSession`, в которой запущена криптографическая операция, в массив байтов `pOperationState`. Размер массива передается через `pulOperationStateLen`. Эта функция удовлетворяет соглашению МНД.

Сохранённое состояние может быть восстановлено с помощью функции `C_SetOperationState`.

Если состояние сессии не может быть сохранено по причине неподдерживаемой криптографической операции или механизма, то будет возвращена ошибка `CKR_STATE_UNSAVEABLE`.

Если криптографическая операция не была запущена перед вызовом данной функции, то будет возвращена ошибка `CKR_OPERATION_NOT_INITIALIZED`.

Коды возврата: `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_STATE_UNSAVEABLE`, `CKR_ARGUMENTS_BAD`.

Пример: см. раздел C_SetOperationState (на стр. 51).

C_SetOperationState

```
CK_DEFINE_FUNCTION(CK_RV, C_SetOperationState)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR       pOperationState,
    CK_ULONG          ulOperationStateLen,
    CK_OBJECT_HANDLE  hEncryptionKey,
    CK_OBJECT_HANDLE  hAuthenticationKey
);
```

Функция C_SetOperationState восстанавливает сохраненное состояние сессии из массива байтов, полученных в результате выполнения C_GetOperationState.

- hSession — состояние хэндл-сессии, состояние которой будет перезаписано;
- pOperationState — массив байтов, полученный в результате выполнения C_GetOperationState;
- ulOperationStateLen — длина массива.

Аргументы hEncryptionKey и hAuthenticationKey должны быть равны CK_INVALID_HANDLE.

Сохраненное состояние не обязательно должно быть получено из той же сессии, в которую происходит восстановление, однако исходная и конечная сессия должны иметь одинаковое состояние (например, CKS_RW_USER_FUNCTIONS), а также должны быть открыты на одном и том же токене.

Сохраненное состояние остается валидным только в рамках одного C_Initaliaze\C_Finalize.

Коды возврата: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_CHANGED, CKR_KEY_NEEDED, CKR_KEY_NOT_NEEDED, CKR_OK, CKR_SAVED_STATE_INVALID, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_ARGUMENTS_BAD.

Пример:

```
CK_SESSION_HANDLE hSession;

CK_OBJECT_HANDLE hKey;

CK_BYTE data1[] = {...};

CK_BYTE data2[] = {...};

CK_BYTE* pSessionState;

CK_ULONG ulSessionStateLen;

CK_BYTE digest[32];

CK_ULONG ulDigestLen;

CK_MECHANISM mechanism = {CKM_GOSTR3411, NULL, 0};
```

```
CK_RV rv;
.
.
rv = C_DigestInit(hSession, &mechanism);
if (rv != CKR_OK) {
.
.
}
rv = C_DigestUpdate(hSession, data1, sizeof(data1));
if (rv != CKR_OK) {
.
.
}
ulSessionStateLen = 0;
rv = C_GetOperationState(hSession, NULL, &ulSessionStateLen);
if (rv != CKR_OK) {
.
.
}
pSessionState = (CK_BYTE_PTR) malloc(ulSessionStateLen);
rv = C_GetOperationState(hSession, pSessionState, &ulSessionStateLen);
if (rv != CKR_OK) {
.
.
}
ulDigestLen = sizeof(digest);
rv = C_DigestFinal(hSession, digest, &ulDigestLen);
if (rv != CKR_OK) {
.
.
}

.
.
```

```
rv = C_SetOperationState(hSession, CK_INVALID_HANDLE, CK_INVALID_HANDLE,
pSessionState, &ulSessionStateLen);

if (rv != CKR_OK) {

.

.

}

rv = C_DigestUpdate(hSession, data2, sizeof(data2));

if (rv != CKR_OK) {

.

.

}

ulDigestLen = sizeof(digest);

rv = C_DigestFinal(hSession, digest, &ulDigestLen);

if (rv != CKR_OK) {

.

.

}
```

Функции создания, изменения и мониторинга объектов

Список функций создания, изменения и мониторинга объектов:

- `C_CreateObject` (на стр. 54);
- `C_CopyObject` (на стр. 56);
- `C_DestroyObject` (на стр. 57);
- `C_GetObjectSize` (на стр. 58);
- `C_GetAttributeValue` (на стр. 58);
- `C_SetAttributeValue` (на стр. 59);
- `C_FindObjectsInit` (на стр. 60);
- `C_FindObjects` (на стр. 61);
- `C_FindObjectsFinal` (на стр. 61).

Функции создания, изменения и мониторинга объектов позволяют прикладной программе создавать и изменять различные объекты как в оперативной памяти компьютера (session-объекты), так и в токене (token-объекты).

C_CreateObject

```
CK_DEFINE_FUNCTION(CK_RV, C_CreateObject)(
    CK_SESSION_HANDLE hSession,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount,
    CK_OBJECT_HANDLE_PTR phObject
);
```

Функция `C_CreateObject` создает новый объект в соответствии с заданным во входных параметрах `pTemplate` и `ulCount` шаблоном — набором атрибутов объекта. Здесь `pTemplate` — набор атрибутов, `ulCount` — количество атрибутов в шаблоне.

Вновь созданный объект будет token-объектом тогда и только тогда, когда в шаблоне присутствует атрибут `CKA_TOKEN` и его значение равно `CK_TRUE`. Во всех иных случаях вновь созданный объект будет session-объектом.

Вновь созданный объект будет private-объектом, соответствующим типу объекта (см. «Атрибуты, устанавливаемые по умолчанию» на стр. 104). Значение атрибута `CKA_PRIVATE` в шаблоне имеет приоритет над значением по умолчанию.

Коды возврата: CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_DOMAIN_PARAMS_INVALID, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE
    hData,
    hCertificate,
    hKey;
CK_OBJECT_CLASS
    dataClass = CKO_DATA,
    certificateClass = CKO_CERTIFICATE,
    keyClass = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_RSA;
CK_CHAR application[] = {“My Application”};
CK_BYTE dataValue[] = {...};
CK_BYTE subject[] = {...};
CK_BYTE id[] = {...};
CK_BYTE certificateValue[] = {...};
CK_BYTE modulus[] = {...};
CK_BYTE exponent[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE dataTemplate[] = {
    {CKA_CLASS, &dataClass, sizeof(dataClass)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_APPLICATION, application, sizeof(application)},
    {CKA_VALUE, dataValue, sizeof(dataValue)}
};
CK_ATTRIBUTE certificateTemplate[] = {
    {CKA_CLASS, &certificateClass, sizeof(certificateClass)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_SUBJECT, subject, sizeof(subject)},
    {CKA_ID, id, sizeof(id)},
    {CKA_VALUE, certificateValue, sizeof(certificateValue)}
};
CK_ATTRIBUTE keyTemplate[] = {
    {CKA_CLASS, &keyClass, sizeof(keyClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_WRAP, &true, sizeof(true)},
    {CKA_MODULUS, modulus, sizeof(modulus)},
    {CKA_PUBLIC_EXPONENT, exponent, sizeof(exponent)}
};
CK_RV rv;
.
.
/* Create a data object */
rv = C_CreateObject(hSession, &dataTemplate, 4, &hData);
```

```
if (rv == CKR_OK) {
    .
    .
}
/* Create a certificate object */
rv = C_CreateObject(
    hSession, &certificateTemplate, 5, &hCertificate);
if (rv == CKR_OK) {
    .
    .
}
/* Create an RSA public key object */
rv = C_CreateObject(hSession, &keyTemplate, 5, &hKey);
if (rv == CKR_OK) {
    .
    .
}
```

C_CopyObject

```
CK_DEFINE_FUNCTION(CK_RV, C_CopyObject)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE hObject,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount,
    CK_OBJECT_HANDLE_PTR phNewObject
);
```

Функция `C_CopyObject` создает копию объекта с дескриптором `hObject`. К созданной копии добавляются атрибуты из шаблона `pTemplate`. Значения атрибутов из `pTemplate` могут задавать новые параметры объекта, например, из session-объекта делать token-объект. Преобразования атрибутов разрешены не во всех случаях. Например, если старый объект содержит атрибут `CKA_MODIFIABLE` и его значение равно `FALSE`, то никакие изменения в его копии с помощью шаблона недопустимы.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_ATTRIBUTE_READ_ONLY`, `CKR_ATTRIBUTE_TYPE_INVALID`, `CKR_ATTRIBUTE_VALUE_INVALID`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OBJECT_HANDLE_INVALID`, `CKR_OK`, `CKR_PIN_EXPIRED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SESSION_READ_ONLY`, `CKR_TEMPLATE_INCONSISTENT`, `CKR_TOKEN_WRITE_PROTECTED`, `CKR_USER_NOT_LOGGED_IN`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey, hNewKey;
CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_DES;
CK_BYTE id[] = {...};
CK_BYTE keyValue[] = {...};
```

```
CK_BBOOL false = CK_FALSE;
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE keyTemplate[] = {
    {CKA_CLASS, &keyClass, sizeof(keyClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_TOKEN, &false, sizeof(false)},
    {CKA_ID, id, sizeof(id)},
    {CKA_VALUE, keyValue, sizeof(keyValue)}
};
CK_ATTRIBUTE copyTemplate[] = {
    {CKA_TOKEN, &true, sizeof(true)}
};
CK_RV rv;
.
.
/* Create a DES secret key session object */
rv = C_CreateObject(hSession, &keyTemplate, 5, &hKey);
if (rv == CKR_OK) {
    /* Create a copy which is a token object */
    rv = C_CopyObject(hSession, hKey, &copyTemplate, 1, &hNewKey);
    .
    .
}
```

C_DestroyObject

```
CK_DEFINE_FUNCTION(CK_RV, C_DestroyObject)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE hObject
);
```

Функция `C_DesroyObject` удаляет объект. Если объект был session-объектом, то он удаляется из памяти компьютера, token-объект удаляется из токена.

Параметры:

- `hSession` — дескриптор сессии;
- `hObject` — дескриптор удаляемого объекта.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OBJECT_HANDLE_INVALID`, `CKR_OK`, `CKR_PIN_EXPIRED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SESSION_READ_ONLY`, `CKR_TOKEN_WRITE_PROTECTED`.

Пример: см. раздел C_GetObjectSize (на стр. 58).

C_GetObjectSize

```
CK_DEFINE_FUNCTION(CK_RV, C_GetObjectSize)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE hObject,
    CK_ULONG_PTR pulSize
);
```

Функция `C_GetObjectSize` возвращает примерный размер объекта в байтах.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_INFORMATION_SENSITIVE`, `CKR_OBJECT_HANDLE_INVALID`, `CKR_OK`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hObject;
CK_OBJECT_CLASS dataClass = CKO_DATA;
CK_CHAR application[] = {“My Application”};
CK_BYTE dataValue[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &dataClass, sizeof(dataClass)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_APPLICATION, application, sizeof(application)},
    {CKA_VALUE, value, sizeof(value)}
};
CK_ULONG ulSize;
CK_RV rv;
.
.
rv = C_CreateObject(hSession, &template, 4, &hObject);
if (rv == CKR_OK) {
    rv = C_GetObjectSize(hSession, hObject, &ulSize);
    if (rv != CKR_INFORMATION_SENSITIVE) {
        .
        .
    }
    rv = C_DestroyObject(hSession, hObject);
    .
    .
}
```

C_GetAttributeValue

```
CK_DEFINE_FUNCTION(CK_RV, C_GetAttributeValue)(
    CK_SESSION_HANDLE hSession,
```

```
    CK_OBJECT_HANDLE hObject,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Функция `C_GetAttributeValue` позволяет прикладной программе получать значения некоторых атрибутов объекта.

Выдача функцией значений атрибутов осуществляется в соответствии с заданными шаблонами — параметры `pTemplate` и `ulCount`. Эта функция удовлетворяет соглашению МНД следующим образом: соглашение применимо к каждому атрибуту, входящему в шаблон.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_ATTRIBUTE_SENSITIVE`, `CKR_ATTRIBUTE_TYPE_INVALID`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OBJECT_HANDLE_INVALID`, `CKR_OK`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hObject;
CK_BYTE_PTR pModulus, pExponent;
CK_ATTRIBUTE template[] = {
    {CKA_MODULUS, NULL_PTR, 0},
    {CKA_PUBLIC_EXPONENT, NULL_PTR, 0}
};
CK_RV rv;
.
.
rv = C_GetAttributeValue(hSession, hObject, &template, 2);
if (rv == CKR_OK) {
    pModulus = (CK_BYTE_PTR) malloc(template[0].ulValueLen);
    template[0].pValue = pModulus;
    /* template[0].ulValueLen was set by C_GetAttributeValue */
    pExponent = (CK_BYTE_PTR) malloc(template[1].ulValueLen);
    template[1].pValue = pExponent;
    /* template[1].ulValueLen was set by C_GetAttributeValue */
    rv = C_GetAttributeValue(hSession, hObject, &template, 2);
    if (rv == CKR_OK) {
        .
        .
    }
    free(pModulus);
    free(pExponent);
}
```

C_SetAttributeValue

```
CK_DEFINE_FUNCTION(CK_RV, C_SetAttributeValue)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE hObject,
```

```
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Функция `C_SetAttributeValue` устанавливает объекту некоторые новые атрибуты в соответствии с заданным шаблоном `pTemplate` и его размером `ulCount`.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_ATTRIBUTE_READ_ONLY`, `CKR_ATTRIBUTE_TYPE_INVALID`, `CKR_ATTRIBUTE_VALUE_INVALID`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OBJECT_HANDLE_INVALID`, `CKR_OK`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SESSION_READ_ONLY`, `CKR_TEMPLATE_INCONSISTENT`, `CKR_TOKEN_WRITE_PROTECTED`, `CKR_USER_NOT_LOGGED_IN`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hObject;
CK_UTF8CHAR label[] = {"New label"};
CK_ATTRIBUTE template[] = {
    CKA_LABEL, label, sizeof(label)-1
};
CK_RV rv;
.
.
rv = C_SetAttributeValue(hSession, hObject, &template, 1);
if (rv == CKR_OK) {
    .
    .
}
```

C_FindObjectsInit

```
CK_DEFINE_FUNCTION(CK_RV, C_FindObjectsInit)(
    CK_SESSION_HANDLE hSession,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

Функция `C_FindObjectsInit` инициализирует процедуру поиска объектов в соответствии с заданным шаблоном `pTemplate` и его размером `ulCount`.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_ATTRIBUTE_TYPE_INVALID`, `CKR_ATTRIBUTE_VALUE_INVALID`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_ACTIVE`, `CKR_PIN_EXPIRED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

Пример: см. раздел C_FindObjectsFinal (на стр. 61).

C_FindObjects

```
CK_DEFINE_FUNCTION(CK_RV, C_FindObjects)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE_PTR phObject,
    CK_ULONG ulMaxObjectCount,
    CK_ULONG_PTR pulObjectCount
);
```

Функция `C_FindObjects` осуществляет поиск объектов в соответствии с шаблонами, заданными в процедуре `C_FindObjectsInit` (на стр. 60). Дескрипторы найденных объектов записываются в массив `phObjects`, максимальный размер которого задается параметром `ulMaxObjectCount`. Число записанных в `phObject` объектов возвращается с помощью указателя `pulObjectCount`.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

Пример: см. раздел C_FindObjectsFinal (на стр. 61).

C_FindObjectsFinal

```
CK_DEFINE_FUNCTION(CK_RV, C_FindObjectsFinal)(
    CK_SESSION_HANDLE hSession
);
```

Функция `C_FindObjectsFinal` завершает процедуру поиска объектов и освобождает использовавшиеся при поиске ресурсы.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

Пример:

```
CK_OBJECT_HANDLE hObject;
CK_ULONG ulObjectCount;
CK_RV rv;
.
.
rv = C_FindObjectsInit(hSession, NULL_PTR, 0);
assert(rv == CKR_OK);
while (1) {
    rv = C_FindObjects(hSession, &hObject, 1, &ulObjectCount);
    if (rv != CKR_OK || ulObjectCount == 0)
        break;
    .
    .
}
```

```
rv = C_FindObjectsFinal(hSession);
assert(rv == CKR_OK);
```

Функции выработки ключей и случайных чисел

Список функций выработки ключей и случайных чисел:

- `C_GenerateRandom` (на стр. 63);
- `C_GenerateKey` (на стр. 64);
- `C_GenerateKeyPair` (на стр. 67);
- `C_DeriveKey` (на стр. 69);
- `C_WrapKey` (на стр. 75);
- `C_UnwrapKey` (на стр. 78).

Функции выработки ключей и случайных чисел позволяют прикладной программе вырабатывать уникальные криптографические параметры, необходимые для шифрования и электронной подписи.

C_GenerateRandom

```
CK_DEFINE_FUNCTION(CK_RV, C_GenerateRandom)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pRandomData,
    CK_ULONG ulRandomLen
);
```

В ViPNet PKCS#11 VT функция `C_GenerateRandom` вырабатывает случайное число в соответствии с настройками датчика случайных чисел, заданными в программе ViPNet CSP (см. документ «ViPNet CSP. Руководство пользователя», раздел «Использование датчика случайных чисел»).

Параметры:

- `pRandomData` — буфер под случайные числа;
- `ulRandomLen` — длина буфера.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_ACTIVE`, `CKR_RANDOM_NO_RNG`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_BYTE seed[] = {...};
CK_BYTE randomData[] = {...};
```

```
CK_RV rv;
.
.
rv = C_GenerateRandom(hSession, randomData, sizeof(randomData));
if (rv == CKR_OK) {
     .
     .
}
```

C_GenerateKey

```
CK_DEFINE_FUNCTION(CK_RV, C_GenerateKey)(
     CK_SESSION_HANDLE hSession
     CK_MECHANISM_PTR pMechanism,
     CK_ATTRIBUTE_PTR pTemplate,
     CK_ULONG ulCount,
     CK_OBJECT_HANDLE_PTR phKey
);
```

В ViPNet PKCS#11 VT функция `C_GenerateKey` осуществляет выработку случайных симметричных ключей типа CKK_GOST28147.

Эта функция поддерживает следующие механизмы:

- `CKM_GOST28147_KEY_GEN`;
- `CKM_PKCS5_PBKD2CKM_GOSTPBE`;
- `CKM_GOSTPBE_94`;
- `CKM_TLS_GOST_PRE_MASTER_KEY_GEN`.

Параметры `pTemplate` и `ulCount` позволяют задавать выработанным ключам те или иные атрибуты.

Примечание. После выработки случайного секретного ключа прикладная программа имеет возможность установить необходимое ей значение секретного ключа с помощью функции `C_SetAttributeValue` (на стр. 59), если в качестве шаблона задать только один атрибут `CKA_VALUE` со значением ключа длиной 32 байта.

Выходным параметром является `phKey` — указатель на дескриптор выработанного ключа.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_ATTRIBUTE_READ_ONLY`, `CKR_ATTRIBUTE_TYPE_INVALID`, `CKR_ATTRIBUTE_VALUE_INVALID`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_MECHANISM_INVALID`, `CKR_MECHANISM_PARAM_INVALID`, `CKR_OK`, `CKR_OPERATION_ACTIVE`, `CKR_PIN_EXPIRED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SESSION_READ_ONLY`, `CKR_TEMPLATE_INCOMPLETE`, `CKR_TEMPLATE_INCONSISTENT`, `CKR_TOKEN_WRITE_PROTECTED`, `CKR_USER_NOT_LOGGED_IN`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_MECHANISM mechanism = {
    CKM_GOST28147_KEY_GEN, NULL_PTR, 0
};
CK_RV rv;
.
.
rv = C_GenerateKey(hSession, &mechanism, NULL_PTR, 0, &hKey);
if (rv == CKR_OK) {
    .
    .
}
```

Механизм CKM_GOST28147_KEY_GEN

Выработка секретного ключа по алгоритму ГОСТ 28147-89.

Не имеет параметров.

Механизм CKM_GOSTPBE_94

Выработка секретного ключа ГОСТ 28147-89 из пароля в соответствии с описанием генерации ключа для схемы PBES2 в документе ТК26.

В качестве механизм принимает указатель на структуру следующего типа:

```
typedef struct CK_PBE_PARAMS {
    CK_BYTE_PTR      pInitVector;
    CK_UTF8CHAR_PTR  pPassword;
    CK_ULONG         ulPasswordLen;
    CK_BYTE_PTR      pSalt;
    CK_ULONG         ulSaltLen;
    CK_ULONG         ulIteration;
} CK_PBE_PARAMS;
```

Где:

- `pInitVector` — должно быть установлено значение `NULL`,
- `pPassword` — пароль, из которого происходит выработка ключа,
- `ulPasswordLen` — длина пароля,
- `pSalt` — массив байтов, подмешиваемая «соль»,
- `ulSaltLen` — размер массива с «солью»,
- `ulIteration` — количество итераций при выработке ключа.

На параметры механизма существуют следующие ограничения:

- `ulSaltLen` должен быть не менее 8 байт и не более 32 байт;
- количество итераций должно быть не менее 1000.

Если параметры не удовлетворяют указанным ограничениям, то функция завершится с ошибкой `CKR_MECHANISM_PARAM_INVALID`.

Механизм CKM_TLS_GOST_PRE_MASTER_KEY_GEN

Выработка секретного ключа для использования в механизме `CKM_TLS_GOST_MASTER_KEY_DERIVE`.

Не имеет параметров.

Механизм CKM_PKCS5_PBKD2

Выработка секретного ключа ГОСТ 28147-89 из пароля в соответствии PKCS#5 PBKDF2.

В качестве параметра механизм принимает указатель на структуру следующего типа:

```
typedef struct CK_PKCS5_PBKD2_PARAMS2 {
    CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE saltSource;
    CK_VOID_PTR pSaltSourceData;
    CK_ULONG ulSaltSourceDataLen;
    CK_ULONG iterations;
    CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf;
    CK_VOID_PTR pPrfData;
    CK_ULONG ulPrfDataLen;
    CK_UTF8CHAR_PTR pPassword;
    CK_ULONG ulPasswordLen;
} CK_PKCS5_PBKD2_PARAMS2;
```

Где:

- `saltSource` — должно быть установлено значение `CKZ_SALT_SPECIFIED`,
- `pSaltSourceData` — массив байтов, подмешиваемая «соль»,
- `ulSaltSourceDataLen` — размер массива с «солью»,
- `iterations` — количество итераций при выработке ключа,
- `prf` — функция псевдослучайной генерации чисел,
- `pPrfData` — параметры для `prf`,
- `ulPrfDataLen` — длина параметров,
- `pPassword` — пароль, из которого происходит выработка ключа,
- `ulPasswordLen` — длина пароля.

Типы поддерживаемых PRF:

- CKP_PKCS5_PBKD2_HMAC_GOSTR3411 — PRF на базе HMAC на основе хэширования по алгоритму ГОСТ Р34.11-94. pPrfData должен иметь значение NULL, ulPrfDataLen должен иметь значение 0.
- CKP_PKCS5_PBKD2_HMAC_GOSTR3411_2012_512 — PRF на базе HMAC на основе хэширования по алгоритму ГОСТ Р34.11-2012 с длиной хэша 512 бит. pPrfData должен иметь значение NULL, ulPrfDataLen должен иметь значение 0.

В pTemplate должен быть указан атрибут CKA_KEY_TYPE со значением CKK_GOST28147.

C_GenerateKeyPair

```
CK_DEFINE_FUNCTION(CK_RV, C_GenerateKeyPair)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_ATTRIBUTE_PTR pPublicKeyTemplate,
    CK_ULONG ulPublicKeyAttributeCount,
    CK_ATTRIBUTE_PTR pPrivateKeyTemplate,
    CK_ULONG ulPrivateKeyAttributeCount,
    CK_OBJECT_HANDLE_PTR phPublicKey,
    CK_OBJECT_HANDLE_PTR phPrivateKey
);
```

Функция C_GenerateKeyPair осуществляет выработку случайных асимметричных ключей, то есть ключевых пар, состоящих из private- и public-ключей. Результатом работы этой функции являются два различных объекта: объект private key и объект public key.

В ViPNet PKCS#11 VT эта функция вырабатывает ключи типа CKK_GOSTR3410 и CKK_GOSTR3410_512 и поддерживает механизмы CKM_GOSTR3410_KEY_PAIR_GEN и CKM_GOSTR3410_512_KEY_PAIR_GEN. Эти механизмы не имеют параметров, значения pParam и ulParamLen должны быть равны NULL_PTR и 0 соответственно.

Параметры pPublicKeyTemplate, ulPublicKeyAttributeCount, pPrivateKeyTemplate и ulPrivateKeyAttributeCount позволяют задавать выработанным ключам те или иные атрибуты.

Внимание! ViPNet PKCS#11 VT не различает ключи ГОСТ Р 34.10-2001 и ГОСТ Р 34.10-2012 с длиной открытого ключа 512 бит.

Единственный способ идентификации алгоритма ключевой пары извне — идентификация по OID параметров хэширования (CKA_GOSTR3411_PARAMS), которые устанавливаются в public-ключ: по стандарту PKCS#11 при генерации ключевой пары ГОСТ Р 34.10-2001 должен быть указан атрибут CKA_GOSTR3411_PARAMS в шаблоне public-ключа. Если же он не установлен, то считается, что это ключевая пара ГОСТ Р 34.10-2012 с длиной открытого ключа 512 бит, а при генерации ключевой пары в атрибут CKA_GOSTR3411_PARAMS будет выставлен OID алгоритма хэширования ГОСТ Р 34.10-2012 с длиной хэша 256 бит.

Выходными параметрами являются указатели `phPublicKey` и `phPrivateKey` на дескрипторы открытого и закрытого ключей соответственно.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_ATTRIBUTE_READ_ONLY`, `CKR_ATTRIBUTE_TYPE_INVALID`, `CKR_ATTRIBUTE_VALUE_INVALID`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_DOMAIN_PARAMS_INVALID`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_MECHANISM_INVALID`, `CKR_MECHANISM_PARAM_INVALID`, `CKR_OK`, `CKR_OPERATION_ACTIVE`, `CKR_PIN_EXPIRED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SESSION_READ_ONLY`, `CKR_TEMPLATE_INCOMPLETE`, `CKR_TEMPLATE_INCONSISTENT`, `CKR_TOKEN_WRITE_PROTECTED`, `CKR_USER_NOT_LOGGED_IN`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hPublicKey, hPrivateKey;
CK_MECHANISM mechanism = {
    CKM_GOST3410_KEY_PAIR_GEN, NULL_PTR, 0
};
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE publicKeyTemplate[] = {
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_VERIFY, &true, sizeof(true)},
    {CKA_WRAP, &true, sizeof(true)},
    {CKA_ID, id, sizeof(id)}
};
CK_ATTRIBUTE privateKeyTemplate[] = {
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_PRIVATE, &true, sizeof(true)},
    {CKA_SUBJECT, subject, sizeof(subject)},
    {CKA_ID, id, sizeof(id)},
    {CKA_SENSITIVE, &true, sizeof(true)},
    {CKA_DECRYPT, &true, sizeof(true)},
    {CKA_SIGN, &true, sizeof(true)},
    {CKA_UNWRAP, &true, sizeof(true)}
};
CK_RV rv;
rv = C_GenerateKeyPair(
    hSession, &mechanism,
    publicKeyTemplate, 4,
    privateKeyTemplate, 8,
    &hPublicKey, &hPrivateKey);
if (rv == CKR_OK) {
    .
    .
}
```

C_DeriveKey

```
CK_DEFINE_FUNCTION(CK_RV, C_DeriveKey)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hBaseKey,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulAttributeCount,
    CK_OBJECT_HANDLE_PTR phKey
);
```

Функция `C_DeriveKey` позволяет строить некоторый производный ключ из базового ключа и (возможно) некоторых параметров механизма.

Входными параметрами являются:

- `hSession` — дескриптор сессии;
- `pMechanism` — указатель на механизм;
- `hBaseKey` — дескриптор базового ключа;
- `pTemplate` — указатель на список атрибутов производного ключа;
- `ulAttributeCount` — количество атрибутов в списке.

Выходным параметром является `phKey` — указатель на дескриптор созданного производного ключа.

В ViPNet PKCS#11 VT функция `C_DeriveKey` поддерживает следующие механизмы:

- `CKM_GOSTR3410_DERIVE` (см. «Механизм CKM_GOSTR3410_DERIVE» на стр. 70);
- `CKM_GOSTR3410_12_DERIVE` (см. «Механизм CKM_GOSTR3410_12_DERIVE» на стр. 71);
- `CKM_GOSTR3410_PUBLIC_KEY_DERIVE` (см. «Механизм CKM_GOSTR3410_PUBLIC_KEY_DERIVE» на стр. 72);
- `CKM_GOSTR3410_512_PUBLIC_KEY_DERIVE` (см. «Механизм CKM_GOSTR3410_512_PUBLIC_KEY_DERIVE» на стр. 72);
- `CKM_TLS_GOST_PRF` (см. «Механизм CKM_TLS_GOST_PRF» на стр. 72);
- `CKM_TLS_GOST_MASTER_KEY_DERIVE` (см. «Механизм CKM_TLS_GOST_MASTER_KEY_DERIVE» на стр. 73);
- `CKM_TLS_GOST_KEY_AND_MAC_DERIVE` (см. «Механизм CKM_TLS_GOST_KEY_AND_MAC_DERIVE» на стр. 74);
- `CKM_KDF_4357` (см. «Механизм CKM_KDF_4357» на стр. 74);
- `CKM_KDF_GOSTR3411_2012_256` (см. «Механизм CKM_KDF_GOSTR3411_2012_256» на стр. 75);
- `CKM_KDF_GOSTR3411_EXPORT` (см. «Механизм CKM_KDF_GOSTR3411_EXPORT» на стр. 75).

При выполнении функции `C_DeriveKey` в случае, если аргумент `phKey` не равен `NULL_PTR`, выработанный в результате выполнения функции объект-ключ удовлетворяет следующим

правилам относительно чувствительности и извлекаемости ключа: атрибуты CKA_SENSITIVE и CKA_EXTRACTABLE в шаблоне могут быть указаны как любое из значений CK_TRUE или CK_FALSE. Если они не указаны явно, то им присваивается значение по умолчанию CK_TRUE. Если для атрибута CKA_ALWAYS_SENSITIVE базового ключа было установлено значение CK_FALSE, то выведенный ключ также будет иметь это значение. Если он был CK_TRUE, то выведенный ключ будет иметь значение CKA_ALWAYS_SENSITIVE, равное значению CKA_SENSITIVE. Аналогично, если атрибут CKA_NEVER_EXTRACTABLE базового ключа был установлен в CK_FALSE, то выведенный ключ также будет иметь это значение. Если он был CK_TRUE, то выведенный ключ будет иметь значение CKA_NEVER_EXTRACTABLE, противоположное значению CKA_EXTRACTABLE.

Коды возврата: CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_DOMAIN_PARAMS_INVALID, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

Механизм CKM_GOSTR3410_DERIVE

Фактически это механизм выработки общего секретного ключа (обозначающегося в RFC 4357 как KEK — key encryption key) типа CKK_GOST28147 с помощью private- и public- ключей ГОСТ Р 34.10 2001 двух корреспондентов, которым необходимо установить между собой секретную связь.

В качестве hBaseKey здесь используется private-ключ одного из двух корреспондентов, public-ключ передается в качестве параметров pPublicData и ulPublicDataLen структуры CK_GOSTR3410_DERIVE_PARAMS, являющейся параметром этого механизма:

```
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;
```

Помимо открытого ключа, в этой структуре есть еще два параметра, kdf и UKM:

- kdf — это идентификатор используемой диверсификации ключа. Он может принимать два значения: CKD_NULL (диверсификация не производится) или CKD_CPDIVERSIFY_KDF (производится диверсификация по алгоритму КриптоПро, описанному в RFC 4357, раздел 6.5);
- UKM — это аббревиатура, применяемая в RFC 4357 и обозначающая 8-байтовый вектор, использующийся при выработке KEK.

Таким образом, в этом механизме всегда должны быть выполнены следующие соотношения:

- ulPublicDataLen = 64;

- `ulUKMLen` = 8.

Отметим, что все параметры структуры `CK_GOSTR3410_DERIVE_PARAM` являются обязательными. При отсутствии или неправильном значении хотя бы одного из них ViPNet PKCS#11 VT выдает ошибку с кодом `CKR_MECHANISM_PARAM_INVALID`.

Объект `hBaseKey`, использующийся в качестве базового ключа, должен содержать атрибут `CKA_KEY_TYPE` со значением `CKK_GOSTR3410`, в противном случае функция возвращает ошибку `CKR_KEY_TYPE_INCONSISTENT`.

Кроме того, объект `hBaseKey` должен содержать атрибут `CKA_DERIVE` со значением `CK_TRUE`, в противном случае функция выдает ошибку с тем же кодом.

Объект `hBaseKey` может содержать атрибут `CKA_GOST28147_PARAMS` со значением OID алгоритма симметричного шифрования. Если такой OID поддерживается, то в случае проведения диверсификации в ней будет использоваться набор параметров симметричного ключа в соответствии с данным OID. Если атрибута нет — значения по умолчанию.

Объект `hBaseKey` может содержать атрибут `CKA_GOSTR3410_PARAMS` со значением OID асимметричных параметров. Если такой OID поддерживается, то при выработке KEK будут использоваться эти параметры. Если атрибута нет — значения по умолчанию.

При выработке KEK при хэшировании всегда используются параметры по умолчанию. Выработанный KEK удовлетворяет СЗСК.

Механизм CKM_GOSTR3410_12_DERIVE

Это механизм выработки общего секретного ключа (обозначающегося в RFC 4357 как KEK — key encryption key) типа CKK_GOST28147 с помощью private- и public-ключей ГОСТ Р 34.10-2012 двух корреспондентов, которым необходимо установить между собой секретную связь.

В качестве `hBaseKey` здесь используется private-ключ одного из двух корреспондентов, public-ключ передается в параметрах этого механизма.

Параметры этого механизма представляют собой байтовый массив следующего формата (в данном случае под операцией | подразумевается присоединение одного массива к другому):

```
CK_BYTE mechParam[] = kdf | publicKeyLength | publicKey | ukmSize | ukm;
```

Где:

- `kdf`, `publicKeyLength` и `ukmSize` — байтовые массивы размера `sizeof(uint_32t)`,
- `publicKey` — байтовый массив размера `publicKeyLength`,
- `ukm` — аббревиатура, применяемая в RFC 4357 и обозначающая 8-байтовый вектор, использующийся при выработке KEK.

Таким образом, в этом механизме всегда должны быть выполнены следующие соотношения:

- `publicKeyLength = 64` для ключевой пары с размером открытого ключа 512 бит;
- `publicKeyLength = 128` для ключевой пары с размером открытого ключа 1024 бит;

- `ukmSize = 8`.

При несоответствии массива описанию выше ViPNet PKCS#11 VT выдает ошибку с кодом `CKR_MECHANISM_PARAM_INVALID`.

Объект `hBaseKey`, использующийся в качестве базового ключа, должен содержать атрибут `CKA_KEY_TYPE` со значением `CKK_GOSTR3410` или `CKK_GOSTR3410_512`, в противном случае функция возвращает ошибку `CKR_KEY_TYPE_INCONSISTENT`.

Кроме того, объект `hBaseKey` должен содержать атрибут `CKA_DERIVE` со значением `CK_TRUE`, в противном случае функция выдает ошибку с тем же кодом.

Объект `hBaseKey` может содержать атрибут `CKA_GOST28147_PARAMS` со значением OID алгоритма симметричного шифрования. Если такой OID поддерживается, то в случае проведения диверсификации в ней будет использоваться набор параметров симметричного ключа в соответствии с данным OID. Если атрибута нет — значения по умолчанию.

Объект `hBaseKey` может содержать атрибут `CKA_GOSTR3410_PARAMS` со значением OID асимметричных параметров. Если такой OID поддерживается, то при выработке KEK будут использоваться эти параметры. Если атрибута нет — значения по умолчанию.

При выработке KEK при хэшировании всегда используются параметры по умолчанию. Выработанный KEK удовлетворяет СЗСК.

Механизм CKM_GOSTR3410_PUBLIC_KEY_DERIVE

Это механизм выработки public-ключа типа `CKK_GOSTR3410` с помощью private-ключа ГОСТ Р 34.10-2001.

Этот механизм не имеет параметров, поэтому `pParameter` должен быть равен `NULL_PTR`, а `ulParameterLen` равен 0.

Механизм CKM_GOSTR3410_512_PUBLIC_KEY_DERIVE

Это механизм выработки public-ключа типа `CKK_GOSTR3410_512` с помощью private-ключа ГОСТ Р 34.10-2012.

Этот механизм не имеет параметров, поэтому `pParameter` должен быть равен `NULL_PTR`, а `ulParameterLen` равен 0.

Механизм CKM_TLS_GOST_PRF

Это алгоритм выработки случайных чисел с помощью базового ключа `hBaseKey`, входной последовательности, именуемой также «затравкой», и некоторой метки. Параметром механизма является структура `CK_TLS_GOST_PRF_PARAMS`:

```
typedef struct CK_TLS_GOST_PRF_PARAMS {
CK_TLS_PRF_PARAMS TlsPrfParams;
```

```
CK_BYTE_PTR pHashParamsOID;
CK_ULONG ulHashParamsOIDLen;
} CK_TLS_GOST_PRF_PARAMS;
```

Здесь `CK_TLS_PRF_PARAMS` — стандартная структура вида:

```
/* CK_TLS_PRF_PARAMS is new for version 2.20 */
typedef struct CK_TLS_PRF_PARAMS {
    CK_BYTE_PTR  pSeed;
    CK_ULONG     ulSeedLen;
    CK_BYTE_PTR  pLabel;
    CK_ULONG     ulLabelLen;
    CK_BYTE_PTR  pOutput;
    CK_ULONG_PTR pulOutputLen;
} CK_TLS_PRF_PARAMS;
```

Таким образом, от стандартной структуры `CK_TLS_PRF_PARAMS` структура `CK_TLS_GOST_PRF_PARAMS` отличается только наличием параметра, определяющего OID процедуры хэширования. Базовый ключ `hBaseKey` должен быть типа `CKK_GENERIC_SECRET` или `CKK_GOST28147` и содержать атрибут `CKA_DERIVE` со значением `TRUE`, в противном случае функция выдает код ошибки `CKR_KEY_FUNCTION_NOT_PERMITTED`. Аргумент `phKey` в этом случае не используется и должен быть равен `NULL_PTR`.

Механизм CKM_TLS_GOST_MASTER_KEY_DERIVE

Это алгоритм выработки мастер-ключа TLS с помощью базового ключа `hBaseKey` и случайных данных клиента и сервера. Параметром механизма является структура `CK_TLS_GOST_MASTER_KEY_DERIVE_PARAMS`:

```
typedef struct CK_TLS_GOST_MASTER_KEY_DERIVE_PARAMS {
CK_SSL3_RANDOM_DATA RandomInfo;
CK_BYTE_PTR pHashParamsOID;
CK_ULONG ulHashParamsOIDLen;
} CK_TLS_GOST_MASTER_KEY_DERIVE_PARAMS;
```

Здесь `CK_SSL3_RANDOM_DATA` — стандартная структура вида:

```
typedef struct CK_SSL3_RANDOM_DATA {
    CK_BYTE_PTR  pClientRandom;
    CK_ULONG     ulClientRandomLen;
    CK_BYTE_PTR  pServerRandom;
    CK_ULONG     ulServerRandomLen;
} CK_SSL3_RANDOM_DATA;
```

Таким образом, от стандартной структуры `CK_SSL3_RANDOM_DATA` структура `CK_TLS_GOST_MASTER_KEY_DERIVE_PARAMS` отличается только наличием параметра, определяющего OID процедуры хэширования. Базовый ключ `hBaseKey` должен быть типа `CKK_GENERIC_SECRET` или `CKK_GOST28147` и содержать атрибут `CKA_DERIVE` со значением `TRUE`, в противном случае функция выдает код ошибки `CKR_KEY_FUNCTION_NOT_PERMITTED`. Вырабатываемый функцией мастер-ключ удовлетворяет соглашению C3CK.

Механизм CKM_TLS_GOST_KEY_AND_MAC_DERIVE

Это алгоритм выработки комплекта ключей (для сервера и клиента) при организации TLS. Параметром механизма является структура `CK_TLS_GOST_KEY_MAT_PARAMS`:

```
typedef struct CK_TLS_GOST_KEY_MAT_PARAMS {
CK_SSL3_KEY_MAT_PARAMS KeyMatParams;
CK_BYTE_PTR pHashParamsOID;
CK_ULONG ulHashParamsOIDLen;
} CK_TLS_GOST_KEY_MAT_PARAMS;
```

Здесь `CK_SSL3_KEY_MAT_PARAMS` — стандартная структура вида:

```
typedef struct CK_SSL3_KEY_MAT_PARAMS {
    CK_ULONG                ulMacSizeInBits;
    CK_ULONG                ulKeySizeInBits;
    CK_ULONG                ulIVSizeInBits;
    CK_BBOOL                bIsExport;
    CK_SSL3_RANDOM_DATA     RandomInfo;
    CK_SSL3_KEY_MAT_OUT_PTR pReturnedKeyMaterial;
} CK_SSL3_KEY_MAT_PARAMS;
```

Таким образом, от стандартной структуры `CK_SSL3_KEY_MAT_PARAMS` структура `CK_TLS_GOST_KEY_MAT_PARAMS` отличается только наличием параметра, определяющего OID процедуры хэширования. Базовый ключ `hBaseKey` должен быть типа `CKK_GENERIC_SECRET` и содержать атрибут `CKA_DERIVE` со значением `TRUE`, в противном случае функция выдает код ошибки `CKR_KEY_FUNCTION_NOT_PERMITTED`.

В стандартной структуре `CK_SSL3_KEY_MAT_PARAMS` есть указатель на структуру `CK_SSL3_KEY_MAT_OUT_PTR` вида:

```
typedef struct CK_SSL3_KEY_MAT_OUT {
    CK_OBJECT_HANDLE hClientMacSecret;
    CK_OBJECT_HANDLE hServerMacSecret;
    CK_OBJECT_HANDLE hClientKey;
    CK_OBJECT_HANDLE hServerKey;
    CK_BYTE_PTR      pIVClient;
    CK_BYTE_PTR      pIVServer;
} CK_SSL3_KEY_MAT_OUT;
```

Эта структура получает результат работы функции — 4 симметричных ключа и два IV. Выработанные симметричные ключи удовлетворяют соглашению СЗСК. Аргумент `phKey` в этом случае не используется и должен быть равен `NULL_PTR`.

Механизм CKM_KDF_4357

Механизм получения диверсифицированного симметричного ключа типа `CKK_GOST28147` из существующего по алгоритму, описанному в RFC 4357.

Механизм в качестве параметра принимает массив байтов длиной 8 байт, который представляет собой значение ukm (см. RFC 4357 http://www.ietf.org/rfc/rfc4357.txt). Если параметр не был

передан или имеет длину, отличную от 8 байт, то будет возвращена ошибка с кодом `CKR_MECHANISM_PARAM_INVALID`.

Механизм CKM_KDF_GOSTR3411_2012_256

Механизм получения диверсифицированного симметричного ключа типа `CKK_GOST28147` из существующего по алгоритму, описанному в документе Рекомендации по стандартизации Р 50.1.113-2016 http://www.tc26.ru/standard/rs/%D0%A0%2050.1.113-2016.pdf, раздел 4.4.

Параметры механизма `CKM_KDF_GOSTR3411_2012_256` задаются байтовым массивом, который имеет следующую структуру:

`label | 0x00 | seed`,

где `label` и `seed` — байтовые массивы, заданные в соответствии с документом Рекомендации по стандартизации Р 50.1.113-2016 http://www.tc26.ru/standard/rs/%D0%A0%2050.1.113-2016.pdf, раздел 4.4.

Если параметр не был передан, то будет возвращена ошибка с кодом `CKR_MECHANISM_PARAM_INVALID`.

Механизм CKM_KDF_GOSTR3411_EXPORT

Механизм получения диверсифицированного симметричного ключа типа `CKK_GOST28147` из существующего по алгоритму, описанному в документе Рекомендации по стандартизации Р 50.1.113-2016 http://www.tc26.ru/standard/rs/%D0%A0%2050.1.113-2016.pdf, раздел 4.6.

Механизм в качестве параметра принимает массив байтов, который представляет собой значение `seed` (см. Рекомендации по стандартизации Р 50.1.113-2016 http://www.tc26.ru/standard/rs/%D0%A0%2050.1.113-2016.pdf, раздел 4.6).

Если параметр не был передан, то будет возвращена ошибка с кодом `CKR_MECHANISM_PARAM_INVALID`.

C_WrapKey

```
CK_DEFINE_FUNCTION(CK_RV, C_WrapKey)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hWrappingKey,
    CK_OBJECT_HANDLE hKey,
    CK_BYTE_PTR pWrappedKey,
    CK_ULONG_PTR pulWrappedKeyLen
);
```

Функция `C_WrapKey` осуществляет шифрование ключей.

Входными параметрами являются:

- `hSession` — дескриптор сессии;
- `pMechanism` — указатель на механизм;
- `hWrappingKey` — дескриптор шифрующего ключа;
- `hKey` — дескриптор шифруемого ключа.

Выходными параметрами являются:

- `pWrappedKey` — зашифрованный ключ;
- `pulWrappedKeyLen` — указатель на длину зашифрованного ключа.

Эта функция поддерживает соглашение МНД.

В ViPNet PKCS#11 VT эта функция поддерживает следующие механизмы:

- `CKM_GOST28147_KEY_WRAP` (см. «Механизм CKM_GOST28147_KEY_WRAP» на стр. 76)
- `CKM_GOSTR3410_KEY_WRAP` (см. «Механизм CKM_GOSTR3410_KEY_WRAP» на стр. 77)
- `CKM_MAGMA_WRAP_PKCS8` (см. «Механизм CKM_MAGMA_WRAP_PKCS8» на стр. 77)

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_KEY_HANDLE_INVALID`, `CKR_KEY_NOT_WRAPPABLE`, `CKR_KEY_SIZE_RANGE`, `CKR_KEY_UNEXTRACTABLE`, `CKR_MECHANISM_INVALID`, `CKR_MECHANISM_PARAM_INVALID`, `CKR_OK`, `CKR_OPERATION_ACTIVE`, `CKR_PIN_EXPIRED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`, `CKR_WRAPPING_KEY_HANDLE_INVALID`, `CKR_WRAPPING_KEY_SIZE_RANGE`, `CKR_WRAPPING_KEY_TYPE_INCONSISTENT`.

Механизм CKM_GOST28147_KEY_WRAP

Это механизм шифрования симметричного ключа типа CKK_GOST28147 с помощью симметричного ключа типа CKK_GOST28147. Результатом работы является последовательность длиной 36 байт, где первые 32 байта — зашифрованный симметричный ключ, последние 4 байта — имитовставка (MAC).

Как `hWrappingKey`, так и `hKey` должны быть типа `CKK_GOST28147`, в противном случае функция возвращает ошибку `CKR_WRAPPING_KEY_TYPE_INCONSISTENT` или `CKR_KEY_TYPE_INCONSISTENT` соответственно. Кроме того, ключ `hWrappingKey` должен иметь атрибут `CKA_WRAP` со значением `TRUE`.

Если `hWrappingKey` имеет атрибут `CKA_GOST28147_PARAMS`, то шифрование осуществляется в соответствии с параметрами, задаваемыми этим атрибутом. Параметром этого механизма является 8-байтовый инициализационный вектор выработки имитовставки. Параметр механизма также может отсутствовать, в этом случае используется нулевой инициализационный вектор.

Механизм CKM_GOSTR3410_KEY_WRAP

Механизм шифрования симметричных ключей ГОСТ 28147-89 с использованием симметричного ключа ГОСТ 28147-89, выработанного по протоколу Диффи-Хеллмана из асимметричных ключей ГОСТ Р 34.10-2001. Результатом работы является последовательность байтов, которая содержит в себе DER-закодированную ASN.1-структуру типа `GostR3410-KeyTransport` (см. RFC 4490, раздел 4.2).

`hWrappingKey` должен содержать открытый ключ принимающей стороны типа `CKK_GOSTR3410`, `hKey` должен быть типа `CKK_GOST28147`, в противном случае функция возвращает ошибку `CKR_WRAPPING_KEY_TYPE_INCONSISTENT` или `CKR_KEY_TYPE_INCONSISTENT` соответственно. Кроме того, ключ `hWrappingKey` должен иметь атрибут `CKA_WRAP` со значением `TRUE`.

В качестве параметра механизм принимает указатель на структуру следующего типа:

```
typedef struct CK_GOSTR3410_KEY_WRAP_PARAMS {
    CK_BYTE_PTR      pWrapOID;
    CK_ULONG         ulWrapOIDLen;
    CK_BYTE_PTR      pUKM;
    CK_ULONG         ulUKMLen;
    CK_OBJECT_HANDLE hKey;
} CK_GOSTR3410_KEY_WRAP_PARAMS;
```

Где:

- `pWrapOID` — DER-закодированный идентификатор криптографических параметров ГОСТ 28147-89. Если указатель имеет значение `NULL`, то будут использоваться параметры A;
- `ulWrapOIDLen` — длина `pWrapOID`, должен быть равен 0, если `pWrapOID` равен `NULL`;
- `pUKM` — указатель на 8-байтовый вектор, использующийся при выработке KEK.

  Если вектор во время `C_WrapKey` равен `NULL`, то будет сгенерирован и использован случайный вектор.

  Если вектор во время `C_WrapKey` не равен `NULL`, то вектор будет сравнён с `ukm`, записанным в зашифрованный блоб с ключом.
- `ulUKMLen` — длина `pUKM`; если `pWrapOID` равен `NULL`, должен быть равен `0`.
- `hKey` — хэндл закрытого ключа отправителя для `C_WrapKey`, хэндл закрытого ключа принимающей стороны для `C_UnwrapKey`. Если в `hKey` для `C_WrapKey` был передан `CK_INVALID_HANDLE`, то будет сгенерирована и использована эфемерная ключевая пара.

Механизм CKM_MAGMA_WRAP_PKCS8

Механизм `CKM_MAGMA_WRAP_PKCS8` используется для экспорта и импорта ключей в соответствии с документом технического комитета по стандартизации «Криптографическая защита информации» (ТК 26) «Транспортный ключевой контейнер. Дополнения к PKCS#8 и PKCS#12».

Экспортировать можно ключи CKK_GOSTR3410 и `CKK_GOSTR3410_512`. Ключ `hWrappingKey`, на котором происходит экспорт, должен иметь тип `CKK_GOST28147` и атрибут `CKA_WRAP` со значением

TRUE. Параметром этого механизма является 8-байтовый инициализационный вектор. Параметр механизма также может отсутствовать.

Как описано в документе, формируется «Портфель ключевой информации» (KeyBag), который содержит в себе DER-закодированную ASN.1-структуру, состоящую из значения ключа и идентификатора эллиптических параметров. Сформированная структура шифруется по алгоритму ГОСТ 28147-89.

C_UnwrapKey

```
CK_DEFINE_FUNCTION(CK_RV, C_UnwrapKey)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hUnwrappingKey,
    CK_BYTE_PTR pWrappedKey,
    CK_ULONG ulWrappedKeyLen,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulAttributeCount,
    CK_OBJECT_HANDLE_PTR phKey
);
```

Функция C_UnwrapKey осуществляет расшифрование ключей. Входными параметрами являются:

- hSession — дескриптор сессии;
- pMechanism — указатель на механизм;
- hUnwrappingKey — дескриптор расшифровывающего ключа;
- pWrappedKey, ulWrappedKeyLen — зашифрованный ключ и его длина;
- pTemplate, ulAttributeCount — набор атрибутов, присваиваемых расшифрованному ключу, и их количество.

Выходным параметром является phKey — указатель на дескриптор расшифрованного ключа. Он поддерживает те же механизмы, что и C_WrapKey (на стр. 75).

Механизм CKM_GOST28147_KEY_WRAP

hUnwrappingKey должен быть типа CKK_GOST28147, в противном случае функция возвращает ошибку CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT. Кроме того, ключ hUnwrappingKey должен иметь атрибут CKA_UNWRAP со значением TRUE.

Если hUnwrappingKey имеет атрибут CKA_GOST28147_PARAMS, то, при наличии соответствующего domain-объекта, шифрование осуществляется в соответствии с параметрами, задаваемыми этим атрибутом. Параметром этого механизма является 8-байтовый инициализационный вектор выработки имитовставки. Параметр механизма также может отсутствовать, в этом случае используется нулевой инициализационный вектор.

Механизм CKM_GOSTR3410_KEY_WRAP

Механизм расшифрования симметричных ключей ГОСТ 28147-89 с использованием симметричного ключа ГОСТ 28147-89, выработанного по протоколу Диффи-Хеллмана из асимметричных ключей ГОСТ Р 34.10-2001.

`hWrappingKey` должен быть равен `CK_INVALID_HANDLE`.

В качестве параметра механизм использует структуру типа `CK_GOSTR3410_KEY_WRAP_PARAMS` (см. «Механизм CKM_GOSTR3410_KEY_WRAP» на стр. 77).

Механизм CKM_MAGMA_WRAP_PKCS8

Механизм импорта асимметричных ключей с использованием симметричного ключа по алгоритму ГОСТ 28147-89. Импортировать можно ключи `CKK_GOSTR3410` и `CKK_GOSTR3410_512`. Ключ, на котором происходит импорт, должен иметь тип `CKK_GOST28147` и атрибут `CKA_UNWRAP` со значением `TRUE`.

Функции шифрования

Список функций шифрования:

- `C_EncryptInit` (на стр. 80);
- `C_EncryptUpdate` (на стр. 81);
- `C_EncryptFinal` (на стр. 82);
- `C_Encrypt` (на стр. 83);
- `C_DecryptInit` (на стр. 84);
- `C_DecryptUpdate` (на стр. 84);
- `C_DecryptFinal` (на стр. 85);
- `C_Decrypt` (на стр. 87).

Функции шифрования позволяют внешней программе осуществлять зашифрование и расшифрование произвольной информации в соответствии с российскими стандартами шифрования.

C_EncryptInit

```
CK_DEFINE_FUNCTION(CK_RV, C_EncryptInit)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey
);
```

Функция `C_EncryptInit` осуществляет инициализацию процедуры шифрования.

Входными параметрами являются:

- `hSession` — дескриптор сессии;
- `pMechanism` — указатель на механизм;
- `hKey` — дескриптор ключа для шифрования.

Выходных параметров нет.

В качестве механизмов могут использоваться:

- `CKM_GOST28147_ECB`;
- `CKM_GOST28147`.

Механизм `CKM_GOST28147_ECB` не имеет параметра, значения `pParam` и `ulParamLen` должны быть равны `NULL_PTR` и `0` соответственно.

В качестве параметра механизма CKM_GOST28147 может быть 8-байтовый IV. Если параметр отсутствует, то используется нулевой IV.

В обоих механизмах если объект-ключ имеет атрибут CKA_GOST28147_PARAMS, то используются параметры, определяемые этим атрибутом. Если такого атрибута нет, то используются параметры по умолчанию.

Коды возврата: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_FUNCTION_NOT_PERMITTED, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

C_EncryptUpdate

```
CK_DEFINE_FUNCTION(CK_RV, C_EncryptUpdate)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pPart,
    CK_ULONG ulPartLen,
    CK_BYTE_PTR pEncryptedPart,
    CK_ULONG_PTR pulEncryptedPartLen
);
```

Функция C_EncryptUpdate выполняет зашифрование очередной части текста, состоящего из нескольких частей. Перед вызовом функции C_EncryptUpdate должна быть проведена инициализация процедуры зашифрования с помощью функции C_EncryptInit (на стр. 80).

Входные параметры:

- hSession — дескриптор сессии;
- pPart — открытый текст;
- ulPartLen — длина открытого текста;

Выходные параметры:

- pEncryptedPart — шифротекст;
- pulEncryptedPartLen — указатель на длину шифротекста.

Эта функция поддерживает соглашение МНД.

Коды возврата: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

C_EncryptFinal

```
CK_DEFINE_FUNCTION(CK_RV, C_EncryptFinal)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pLastEncryptedPart,
    CK_ULONG_PTR pulLastEncryptedPartLen
);
```

Функция `C_EncryptFinal` завершает выполнение процедуры зашифрования, возвращает последний зашифрованный блок и освобождает все ресурсы, задействованные в процедуре зашифрования.

Входной параметр `hSession` — дескриптор сессии.

Выходные параметры:

- `pLastEncryptedPart` — последний блок шифротекста;
- `pulLastEncryptedPartLen` — указатель на длину последнего блока шифротекста.

Функция поддерживает соглашение МНД.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DATA_LEN_RANGE`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

Пример:

```
#define PLAINTEXT_BUF_SZ 200
#define CIPHERTEXT_BUF_SZ 256
CK_ULONG firstPieceLen, secondPieceLen;
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_BYTE iv[8];
CK_MECHANISM mechanism = {
    CKM_GOST28147, iv, sizeof(iv)
};
CK_BYTE data[PLAINTEXT_BUF_SZ];
CK_BYTE encryptedData[CIPHERTEXT_BUF_SZ];
CK_ULONG ulEncryptedData1Len;
CK_ULONG ulEncryptedData2Len;
CK_ULONG ulEncryptedData3Len;
CK_RV rv;
.
.
firstPieceLen = 90;
secondPieceLen = PLAINTEXT_BUF_SZ-firstPieceLen;
rv = C_EncryptInit(hSession, &mechanism, hKey);
if (rv == CKR_OK) {
    /* Encrypt first piece */
    ulEncryptedData1Len = sizeof(encryptedData);
    rv = C_EncryptUpdate(
```

```
        hSession,
        &data[0], firstPieceLen,
        &encryptedData[0], &ulEncryptedData1Len);
    if (rv != CKR_OK) {
          .
          .
        }
    /* Encrypt second piece */
    ulEncryptedData2Len = sizeof(encryptedData)-ulEncryptedData1Len;
    rv = C_EncryptUpdate(
      hSession,
      &data[firstPieceLen], secondPieceLen,
      &encryptedData[ulEncryptedData1Len], &ulEncryptedData2Len);
    if (rv != CKR_OK) {
          .
          .
    }
    /* Get last little encrypted bit */
    ulEncryptedData3Len =
      sizeof(encryptedData)-ulEncryptedData1Len-ulEncryptedData2Len;
    rv = C_EncryptFinal(
        hSession,
        &encryptedData[ulEncryptedData1Len+ulEncryptedData2Len],
        &ulEncryptedData3Len);
    if (rv != CKR_OK) {
          .
          .
    }
}
```

C_Encrypt

```
CK_DEFINE_FUNCTION(CK_RV, C_Encrypt)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pData,
    CK_ULONG ulDataLen,
    CK_BYTE_PTR pEncryptedData,
    CK_ULONG_PTR pulEncryptedDataLen
);
```

Функция `C_Encrypt` последовательно выполняет `C_EncryptUpdate` (на стр. 81) и `C_EncryptFinal` (на стр. 82).

Входные параметры:

- `hSession` — дескриптор сессии;
- `pData` — открытый текст;
- `ulDataLen` — длина открытого текста.

Выходные параметры:

- `pEncryptedData` — шифротекст;
- `pulEncryptedDataLen` — указатель на длину шифротекста.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DATA_INVALID`, `CKR_DATA_LEN_RANGE`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

C_DecryptInit

```
CK_DEFINE_FUNCTION(CK_RV, C_DecryptInit)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey
);
```

Функция `C_DecryptInit` инициализирует процедуру расшифрования. Она поддерживает те же механизмы, что и `C_EncryptInit` (на стр. 80).

Входными параметрами являются:

- `hSession` — дескриптор сессии;
- `pMechanism` — указатель на механизм;
- `hKey` — дескриптор ключа для шифрования.

Выходных параметров нет.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_KEY_FUNCTION_NOT_PERMITTED`, `CKR_KEY_HANDLE_INVALID`, `CKR_KEY_SIZE_RANGE`, `CKR_KEY_TYPE_INCONSISTENT`, `CKR_MECHANISM_INVALID`, `CKR_MECHANISM_PARAM_INVALID`, `CKR_OK`, `CKR_OPERATION_ACTIVE`, `CKR_PIN_EXPIRED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`.

C_DecryptUpdate

```
CK_DEFINE_FUNCTION(CK_RV, C_DecryptUpdate)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pEncryptedPart,
    CK_ULONG ulEncryptedPartLen,
    CK_BYTE_PTR pPart,
    CK_ULONG_PTR pulPartLen
);
```

Функция `C_DecryptUpdate` выполняет расшифрование очередной части текста, состоящего из нескольких частей. Перед вызовом функции `C_DecryptUpdate` должна быть проведена инициализация процедуры расшифрования с помощью функции `C_DecryptInit` (на стр. 84).

Входные параметры:

- `hSession` — дескриптор сессии;
- `pEncryptedPart` — шифротекст;
- `ulEncryptedPartLen` — длина шифротекста;

Выходные параметры:

- `pPart` — открытый текст;
- `pulPartLen` — указатель на длину открытого текста.

Эта функция поддерживает соглашение МНД.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_ENCRYPTED_DATA_INVALID`, `CKR_ENCRYPTED_DATA_LEN_RANGE`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`.

C_DecryptFinal

```
      CK_SESSION_HANDLE hSession,
      CK_BYTE_PTR pLastPart,
      CK_ULONG_PTR pulLastPartLen
);
```

Функция `C_DecryptFinal` завершает выполнение процедуры расшифрования, возвращает последний расшифрованный блок и освобождает все ресурсы, задействованные в процедуре расшифрования.

Входной параметр `hSession` — дескриптор сессии.

Выходные параметры:

- `pLastPart` — последний блок открытого текста;
- `pulLastPartLen` — указатель на длину последнего блока открытого текста.

Функция поддерживает соглашение МНД.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_ENCRYPTED_DATA_INVALID`, `CKR_ENCRYPTED_DATA_LEN_RANGE`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`.

Пример:

```
#define CIPHERTEXT_BUF_SZ 256
#define PLAINTEXT_BUF_SZ 256
CK_ULONG firstEncryptedPieceLen, secondEncryptedPieceLen;
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_BYTE iv[8];
CK_MECHANISM mechanism = {
    CKM_GOST28147, iv, sizeof(iv)
};
CK_BYTE data[PLAINTEXT_BUF_SZ];
CK_BYTE encryptedData[CIPHERTEXT_BUF_SZ];
CK_ULONG ulData1Len, ulData2Len, ulData3Len;
CK_RV rv;
.
.
firstEncryptedPieceLen = 90;
secondEncryptedPieceLen = CIPHERTEXT_BUF_SZ-firstEncryptedPieceLen;
rv = C_DecryptInit(hSession, &mechanism, hKey);
if (rv == CKR_OK) {
    /* Decrypt first piece */
    ulData1Len = sizeof(data);
    rv = C_DecryptUpdate(
        hSession,
        &encryptedData[0], firstEncryptedPieceLen,
        &data[0], &ulData1Len);
    if (rv != CKR_OK) {
        .
        .
    }
    /* Decrypt second piece */
    ulData2Len = sizeof(data)-ulData1Len;
    rv = C_DecryptUpdate(
        hSession,
        &encryptedData[firstEncryptedPieceLen],
        secondEncryptedPieceLen,
        &data[ulData1Len], &ulData2Len);
    if (rv != CKR_OK) {
        .
        .
    }
    /* Get last little decrypted bit */
    ulData3Len = sizeof(data)-ulData1Len-ulData2Len;
    rv = C_DecryptFinal(
        hSession,
        &data[ulData1Len+ulData2Len], &ulData3Len);
    if (rv != CKR_OK) {
          .
          .
    }
}
```

C_Decrypt

```
CK_DEFINE_FUNCTION(CK_RV, C_Decrypt)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pEncryptedData,
    CK_ULONG ulEncryptedDataLen,
    CK_BYTE_PTR pData,
    CK_ULONG_PTR pulDataLen
);
```

Функция `C_Decrypt` последовательно выполняет `C_DecryptUpdate` (на стр. 84) и `C_DecryptFinal` (на стр. 85).

Входные параметры:

- `hSession` — дескриптор сессии;
- `pEncryptedData` — шифротекст;
- `ulEncryptedDataLen` — длина шифротекста.

Выходные параметры:

- `pData` — открытый текст;
- `pulDataLen` — указатель на длину открытого текста.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_ENCRYPTED_DATA_INVALID`, `CKR_ENCRYPTED_DATA_LEN_RANGE`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`.

Функции хэширования

Список функций хэширования:

- `C_DigestInit` (на стр. 88);
- `C_DigestUpdate` (на стр. 89);
- `C_DigestKey` (на стр. 89);
- `C_DigestFinal` (на стр. 90);
- `C_Digest` (на стр. 91).

Функции хэширования предназначены для вычисления хэш-функций (digest) от произвольной информации и криптографических ключей.

C_DigestInit

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestInit)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism
);
```

Функция `C_DigestInit` инициализирует процедуру хэширования. Входными параметрами являются:

- `hSession` — дескриптор сессии;
- `pMechanism` — указатель на механизм.

Выходных параметров нет.

Функция поддерживает следующие механизмы:

- `CKM_GOSTR3411`.
- `CKM_GOSTR3411_12_256`.
- `CKM_GOSTR3411_12_512`.

У механизма `CKM_GOSTR3411` может присутствовать DER-закодированный идентификатор криптографических параметров ГОСТ Р 34.11-94, используемых при вычислении хэш-функции. Параметр механизма может отсутствовать, в этом случае будут использованы криптографические параметры по умолчанию.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_MECHANISM_INVALID`, `CKR_MECHANISM_PARAM_INVALID`, `CKR_OK`, `CKR_OPERATION_ACTIVE`, `CKR_PIN_EXPIRED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`.

C_DigestUpdate

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestUpdate)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pPart,
    CK_ULONG ulPartLen
);
```

Функция `C_DigestUpdate` выполняет хэширование очередной части текста, состоящего из нескольких частей. Перед вызовом функции `C_DigestUpdate` должна быть проведена инициализация процедуры хэширования с помощью функции `C_DigestInit` (на стр. 88).

Входные параметры:

- `hSession` — дескриптор сессии;
- `pPart` — хэшируемый текст;
- `ulPartLen` — длина хэшируемого текста.

Выходных параметров нет.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

C_DigestKey

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestKey)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE hKey
);
```

Функция `C_DigestKey` выполняет хэширование очередной порции данных, где в качестве таких данных выступают значения секретного ключа `hKey`. Перед вызовом функции `C_DigestKey` должна быть проведена инициализация процедуры хэширования с помощью функции `C_DigestInit` (на стр. 88).

Входные параметры:

- `hSession` — дескриптор сессии;
- `hKey` — дескриптор хэшируемого ключа.

Выходных параметров нет.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_KEY_HANDLE_INVALID`, `CKR_KEY_INDIGESTIBLE`, `CKR_KEY_SIZE_RANGE`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

C_DigestFinal

```
CK_DEFINE_FUNCTION(CK_RV, C_DigestFinal)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pDigest,
    CK_ULONG_PTR pulDigestLen
);
```

Функция `C_DigestFinal` завершает выполнение процедуры хэширования, возвращает значение хэш-функции и освобождает все ресурсы, задействованные в процедуре хэширования.

Входной параметр `hSession` — дескриптор сессии.

Выходные параметры `pDigest` и `pulDigestLen` — указатели на значение хэш-функции и на его длину соответственно.

Функция поддерживает соглашение МНД.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_MECHANISM mechanism = {
    CKM_GOST3411, NULL_PTR, 0
};
CK_BYTE data[] = {...};
CK_BYTE digest[32];
CK_ULONG ulDigestLen;
CK_RV rv;
.
.
rv = C_DigestInit(hSession, &mechanism);
if (rv != CKR_OK) {
    .
    .
}
rv = C_DigestUpdate(hSession, data, sizeof(data));
if (rv != CKR_OK) {
    .
    .
}
rv = C_DigestKey(hSession, hKey);
if (rv != CKR_OK) {
    .
    .
}
ulDigestLen = sizeof(digest);
rv = C_DigestFinal(hSession, digest, &ulDigestLen);
```

C_Digest

```
CK_DEFINE_FUNCTION(CK_RV, C_Digest)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pData,
    CK_ULONG ulDataLen,
    CK_BYTE_PTR pDigest,
    CK_ULONG_PTR pulDigestLen
);
```

Функция `C_Digest` последовательно выполняет `C_DigestUpdate` (на стр. 89) и `C_DigestFinal` (на стр. 90).

Входные параметры:

- `hSession` — дескриптор сессии;
- `pData` — хэшируемый текст;
- `ulDataLen` — длина хэшируемого текста.

Выходные параметры `pDigest` и `pulDigestLen` — указатели на значение хэш-функции и на его длину соответственно.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

Функции подписи и проверки подписи

Список функций подписи и проверки подписи:

- `C_SignInit` (на стр. 92);
- `C_SignUpdate` (на стр. 93);
- `C_SignFinal` (на стр. 94);
- `C_Sign` (на стр. 95);
- `C_VerifyInit` (на стр. 95);
- `C_VerifyUpdate` (на стр. 96);
- `C_VerifyFinal` (на стр. 97);
- `C_Verify` (на стр. 97).

Функции подписи и проверки подписи позволяют приложению организовать систему подтверждения подлинности сообщения с помощью электронной подписи.

C_SignInit

```
CK_DEFINE_FUNCTION(CK_RV, C_SignInit)(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey
);
```

Функция `C_SignInit` инициализирует процедуру подписи.

Входными параметрами являются:

- `hSession` — дескриптор сессии;
- `pMechanism` — указатель на механизм;
- `hKey` — дескриптор ключа подписи.

Выходных параметров нет.

Функция поддерживает следующие механизмы:

- `CKM_GOST28147_MAC`;
- `CKM_GOSTR3411_HMAC`;
- `CKM_GOSTR3410`;

- CKM_GOSTR3410_WITH_GOSTR3411;
- CKM_GOSTR3410_WITH_GOSTR3411_12_256;
- CKM_GOSTR3410_WITH_GOSTR3411_12_512;
- CKM_GOSTR3410_512;
- CKM_GOSTR3411_12_256_HMAC;
- CKM_GOSTR3411_12_512_HMAC.

Параметром механизма CKM_GOST28147_MAC является 8-байтовый инициализационный вектор. Параметр механизма также может отсутствовать, в этом случае используется нулевой инициализационный вектор.

Параметром механизма CKM_GOSTR3410_WITH_GOSTR3411 является DER-закодированный идентификатор криптографических параметров ГОСТ Р 34.11 94, используемых при вычислении хэш-функции. Параметр механизма может отсутствовать, при этом будут использованы криптографические параметры по умолчанию.

На вход подаются данные, которые хэшируются с помощью ГОСТ Р 34.11-94, а потом подписываются с помощью ГОСТ Р 34.10-2001.

Механизмы CKM_GOSTR3410 и CKM_GOSTR3410_512 не имеют параметра. На вход подается уже посчитанный хэш с помощью механизма ГОСТ Р 34.11-94 и ГОСТ Р 34.11-2012 512 бит соответственно.

Коды возврата: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_FUNCTION_NOT_PERMITTED, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

C_SignUpdate

```
CK_DEFINE_FUNCTION(CK_RV, C_SignUpdate)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pPart,
    CK_ULONG ulPartLen
);
```

Функция C_SignUpdate выполняет включение очередной части текста, состоящего из нескольких частей, в процедуру подписи. Перед вызовом функции C_SignUpdate должна быть проведена инициализация процедуры подписи с помощью функции C_SignInit (на стр. 92).

Входные параметры:

- hSession — дескриптор сессии;
- pPart — хэшируемый текст;

- `ulPartLen` — длина хэшируемого текста.

Выходных параметров нет.

Примечание. Для механизмов `CKM_GOSTR3410` и `CKM_GOSTR3410_512` в качестве `pPart` на вход должно быть подано значение хэш-функции, значение `ulPartLen` должно быть равно 32 и 64 байта соответственно.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DATA_LEN_RANGE`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`.

C_SignFinal

```
CK_DEFINE_FUNCTION(CK_RV, C_SignFinal)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pSignature,
    CK_ULONG_PTR pulSignatureLen
);
```

Функция `C_SignFinal` завершает выполнение процедуры подписи, возвращает значение подписи и освобождает все ресурсы, задействованные в процедуре подписи.

Входной параметр `hSession` — дескриптор сессии.

Выходные параметры `pSignature` и `pulSignatureLen` — указатели на значение подписи и на ее длину соответственно.

Функция поддерживает соглашение МНД.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DATA_LEN_RANGE`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`, `CKR_FUNCTION_REJECTED`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_MECHANISM mechanism = {
    CKM_GOST28147_MAC, NULL_PTR, 0
};
CK_BYTE data[] = {...};
CK_BYTE mac[4];
CK_ULONG ulMacLen;
CK_RV rv;
.
```

```
.
rv = C_SignInit(hSession, &mechanism, hKey);
if (rv == CKR_OK) {
    rv = C_SignUpdate(hSession, data, sizeof(data));
    .
    .
    ulMacLen = sizeof(mac);
    rv = C_SignFinal(hSession, mac, &ulMacLen);
    .
    .
}
```

C_Sign

```
CK_DEFINE_FUNCTION(CK_RV, C_Sign)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pData,
    CK_ULONG ulDataLen,
    CK_BYTE_PTR pSignature,
    CK_ULONG_PTR pulSignatureLen
);
```

Функция `C_Sign` последовательно выполняет `C_SignUpdate` (на стр. 93) и `C_SignFinal` (на стр. 94).

Входные параметры:

- `hSession` — дескриптор сессии;
- `pData` — подписываемый текст;
- `ulDataLen` — длина подписываемого текста.

Выходные параметры `pSignature` и `pulSignatureLen` — указатели на значение подписи и на ее длину соответственно.

Примечание. Для механизмов `CKM_GOSTR3410` и `CKM_GOSTR3410_512` в качестве `pData` на вход должно быть подано готовое значение хэш-функции, значение `ulDataLen` должно быть равно 32 и 64 байта соответственно.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_BUFFER_TOO_SMALL`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DATA_INVALID`, `CKR_DATA_LEN_RANGE`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_USER_NOT_LOGGED_IN`, `CKR_FUNCTION_REJECTED`.

C_VerifyInit

```
CK_DEFINE_FUNCTION(CK_RV, C_VerifyInit)(
```

```
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey
);
```

Инициализирует процедуру проверки подписи.

Входными параметрами являются:

- `hSession` — дескриптор сессии;
- `pMechanism` — указатель на механизм;
- `hKey` — дескриптор ключа для проверки подписи.

Выходных параметров нет.

Поддерживает те же механизмы, что и `C_SignInit` (на стр. 92).

C_VerifyUpdate

```
CK_DEFINE_FUNCTION(CK_RV, C_VerifyUpdate)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pPart,
    CK_ULONG ulPartLen
);
```

Функция `C_VerifyUpdate` выполняет включение очередной части текста, состоящего из нескольких частей, в процедуру проверки подписи. Перед вызовом функции `C_VerifyUpdate` должна быть проведена инициализация процедуры проверки подписи с помощью функции `C_VerifyInit` (на стр. 95).

Входные параметры:

- `hSession` — дескриптор сессии;
- `pPart` — хэшируемый текст;
- `ulPartLen` — длина хэшируемого текста.

Выходных параметров нет.

Примечание. Для механизмов `CKM_GOSTR3410` и `CKM_GOSTR3410_512` в качестве `pPart` на вход должно быть подано готовое значение хэш-функции, значение `ulPartLen` должно быть равно 32 и 64 байта соответственно.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DATA_LEN_RANGE`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`.

C_VerifyFinal

```
CK_DEFINE_FUNCTION(CK_RV, C_VerifyFinal)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pSignature,
    CK_ULONG ulSignatureLen
);
```

Функция `C_VerifyFinal` завершает выполнение процедуры проверки подписи и освобождает все ресурсы, задействованные в этой процедуре.

Входные параметры:

- `hSession` — дескриптор сессии.
- `pSignature` и `ulSignatureLen` — подпись и ее длина соответственно.

Выходных параметров нет.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DATA_LEN_RANGE`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SIGNATURE_INVALID`, `CKR_SIGNATURE_LEN_RANGE`.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_MECHANISM mechanism = {
    CKM_GOST28147_MAC, NULL_PTR, 0
};
CK_BYTE data[] = {...};
CK_BYTE mac[4];
CK_RV rv;
.
.
rv = C_VerifyInit(hSession, &mechanism, hKey);
if (rv == CKR_OK) {
    rv = C_VerifyUpdate(hSession, data, sizeof(data));
    .
    .
    rv = C_VerifyFinal(hSession, mac, sizeof(mac));
    .
    .
}
```

C_Verify

```
CK_DEFINE_FUNCTION(CK_RV, C_Verify)(
    CK_SESSION_HANDLE hSession,
```

```
    CK_BYTE_PTR pData,
    CK_ULONG ulDataLen,
    CK_BYTE_PTR pSignature,
    CK_ULONG ulSignatureLen
);
```

Функция `C_Verify` последовательно выполняет `C_VerifyUpdate` (на стр. 96) и `C_VerifyFinal` (на стр. 97).

Входные параметры:

- `hSession` — дескриптор сессии;
- `pData` — хэшируемый текст;
- `ulDataLen` — длина хэшируемого текста;
- `pSignature` и `ulSignatureLen` — подпись и ее длина соответственно.

Выходных параметров нет.

Примечание. Для механизмов `CKM_GOSTR3410` и `CKM_GOSTR3410_512` в качестве `pData` на вход должно быть подано готовое значение хэш-функции, значение `ulDataLen` должно быть равно 32 и 64 байта соответственно.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_DATA_INVALID`, `CKR_DATA_LEN_RANGE`, `CKR_DEVICE_ERROR`, `CKR_DEVICE_MEMORY`, `CKR_DEVICE_REMOVED`, `CKR_FUNCTION_CANCELED`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`, `CKR_OPERATION_NOT_INITIALIZED`, `CKR_SESSION_CLOSED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_SIGNATURE_INVALID`, `CKR_SIGNATURE_LEN_RANGE`.

Функции расширения интерфейса PKCS#11

Список функций расширения интерфейса PKCS#11 (вспомогательных функций для работы с ViPNet PKCS#11 VT):

- `C_GetFunctionListEx` (на стр. 99);
- `C_AttachToken` (на стр. 100);
- `C_DettachToken` (на стр. 100);
- `C_CreateToken` (на стр. 101);
- `C_RemoveToken` (на стр. 101);
- `C_CheckLicenseInfo` (на стр. 102).

C_GetFunctionListEx

```
CK_RV C_GetFunctionListEx(CK_FUNCTION_LIST_EX_PTR_PTR ppFunctionList)
```

Возвращаемым параметром функции `C_GetFunctionListEx` является указатель списка адресов функций расширенного интерфейса ViPNet PKCS#11 VT.

Использование этой функции значительно облегчает программистам получение необходимых адресов функций из библиотеки `softtoken_pkcs11`.

Параметр `ppFunctionList` является указателем на указатель структуры `CK_FUNCTION_LIST_EX`, которая определена в заголовочном файле `pkcs11_extensions.h`.

Коды возврата: `CKR_ARGUMENTS_BAD`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Пример:

```
CK_FUNCTION_LIST_EX_PTR pFunctionListEx;
CK_C_UpdateTokenList pC_UpdateTokenList;
const char* label = “test label”;
CK_RV rv;
CK_SLOT_ID newSlot = 8888;
/* It’s OK to call C_GetFunctionListEx before calling C_Initialize */
rv = C_GetFunctionListEx(&pFunctionListEx);
assert(rv == CKR_OK);
rv = C_Initialize(NULL);
assert(rv == CKR_OK);
pC_UpdateTokenList = pFunctionList -> C_CreateToken;
/* Call the C_CreateToken() function in the library */
rv = (*C_CreateToken ())(newSlot, (CK_UTF8CHAR_PTR)label, strlen(label));
```

C_AttachToken

```
CK_RV C_AttachToken(CK_SLOT_ID slotID)
```

Эмулирует установку токена в слот.

Параметры:

`slotID` — идентификатор слота с токеном.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_SLOT_ID_INVALID`, `CKR_ARGUMENTS_BAD`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Пример: см. раздел `C_DettachToken` (на стр. 100).

C_DettachToken

```
CK_RV C_DettachToken(CK_SLOT_ID slotID)
```

Эмулирует изъятие токена из слота. Если токен изъят, при попытке открытия сессии на данном номере слота или попытке получения информации о токене будет возвращена ошибка `CKR_TOKEN_NOT_PRESENT`.

Параметры:

`slotID` — идентификатор слота с токеном.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_SLOT_ID_INVALID`, `CKR_ARGUMENTS_BAD`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Пример:

```
CK_SLOT_ID slotID;
CK_SESSION_HANDLE hSession;
CK_RV rv;
.
.
/* Detach token */
rv = C_DettachToken(slotID);
assert(rv == CKR_OK);
/* Can not open session on detached token */
rv = C_OpenSession( slotID, CKF_SERIAL_SESSION | CKF_RW_SESSION, NULL, NULL,
&hSession );
assert(rv == CKR_TOKEN_NOT_PRESENT);
.
.
/* Attach token */
rv = C_AttachToken(slotID);
assert(rv == CKR_OK);
rv = C_OpenSession( slotID, CKF_SERIAL_SESSION | CKF_RW_SESSION, NULL, NULL,
&hSession );
assert(rv == CKR_OK);
```

C_CreateToken

```
CK_RV C_CreateToken (CK_SLOT_ID slotID, CK_UTF8CHAR_PTR creationLabel, CK_ULONG
creationLabelSize)
```

Функция `C_CreateToken` создаёт хранилище-токен.

Параметры:

`slotID` — идентификатор слота с токеном.

Внимание! Вызов `C_CreateToken` из разных потоков или процессов с одинаковым `slotID` приведет к неопределенному поведению.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_SLOT_ID_INVALID`, `CKR_ARGUMENTS_BAD`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Пример: см. раздел C_RemoveToken (на стр. 101).

C_RemoveToken

```
CK_RV C_RemoveToken(CK_SLOT_ID slotID)
```

Функция `C_RemoveToken` удаляет хранилище-токен, соответствующее аргументу `slotID`.

Параметры:

`slotID` — идентификатор слота с токеном.

Внимание! Вызов `C_RemoveToken` из разных потоков или процессов с одинаковым `slotID` приведет к неопределенному поведению.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_SLOT_ID_INVALID`, `CKR_ARGUMENTS_BAD`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Пример:

```
CK_SLOT_ID slotID;
CK_MECHANISM_INFO info;
CK_ULONG ulSlotCount;
CK_RV rv;
.
.
/* Create new slot-token */
rv = C_CreateToken(slotID);
if(rv == CKR_OK)
{
...
```

```
}
/* Remove the token */
rv = C_RemoveToken(slotID);
assert(rv == CKR_OK);
```

C_CheckLicenseInfo

```
CK_RV C_CheckLicenseInfo(CK_VOID_PTR pContext)
```

Функция проверяет лицензию. Параметр `pContext` должен быть равен `NULL_PTR`.

Пример:

```
CK_RV rv;
.
.
/* Create new slot-token */
rv = C_CheckLicenseInfo(NULL);
if (rv != CKR_OK && rv == CKR_INVALID_LICENSE)
{
...
}
```

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_INVALID_LICENSE`, `CKR_OK`.

C_ReglamentInspection

```
CK_RV C_ReglamentInspection(CK_SLOT_ID slotID)
```

Проверка работоспособности. Порядок вызова функции указан в сертификационной документации.

Параметры:

`slotID` — идентификатор слота с токеном.

Коды возврата: `CKR_CRYPTOKI_NOT_INITIALIZED`, `CKR_SESSION_HANDLE_INVALID`, `CKR_FUNCTION_FAILED`.

Функции из стандарта PKCS#11, не поддерживаемые в ViPNet PKCS#11 VT

Список функций, которые не поддерживаются в ViPNet PKCS#11 VT:

- `C_GetFunctionStatus`;
- `C_CancelFunction`;
- `C_DigestEncryptUpdate`;
- `C_DecryptDigestUpdate`;
- `C_SignEncryptUpdate`;
- `C_DecryptVerifyUpdate`;
- `C_SignRecoverInit`;
- `C_SignRecover`;
- `C_VerifyRecoverInit`;
- `C_VerifyRecover`;
- `C_SeedRandom`.

Атрибуты, устанавливаемые по умолчанию

При выполнении некоторых функций интерфейса PKCS#11 происходит создание новых объектов как на токене, так и в оперативной памяти компьютера. Как правило, атрибуты вновь создаваемых объектов задаются внешней программой с помощью шаблонов со списком устанавливаемых атрибутов. Однако в некоторых случаях установку некоторых атрибутов по умолчанию целесообразно осуществлять с помощью самого интерфейса PKCS#11 без помощи внешних шаблонов. Некоторые из таких атрибутов по умолчанию могут быть затем изменены внешней программой с помощью функции `C_SetAttributeValue` (на стр. 59).

Ниже приводится таблица атрибутов, устанавливаемых по умолчанию при выполнении различных функций интерфейса PKCS#11.

Таблица 11. Атрибуты, устанавливаемые по умолчанию

Объект	Функция	Механизм	Атрибуты по умолчанию
Private key	C_GenerateKeyPair	CKM_GOSTR3410_KEY_PAIR_GEN	CKA_CLASS = CKO_PRIVATE_KEY CKA_PRIVATE = CK_TRUE CKA_KEY_TYPE = CKK_GOSTR3410
Public key	C_GenerateKeyPair	CKM_GOSTR3410_KEY_PAIR_GEN	CKA_CLASS = CKO_PUBLIC_KEY CKA_PRIVATE = CK_FALSE CKA_KEY_TYPE = CKK_GOSTR3410
Secret key	C_GenerateKey	CKM_GOST28147_KEY_GEN	CKA_CLASS = CKO_SECRET_KEY CKA_PRIVATE = CK_TRUE CKA_KEY_TYPE = CKK_GOST28147
Secret key	C_UnwrapKey	CKM_GOST28147_KEY_WRAP	CKA_CLASS = CKO_SECRET_KEY CKA_PRIVATE = CK_TRUE CKA_KEY_TYPE = CKK_GOST28147
Secret key	C_DeriveKey	CKM_GOSTR3410_DERIVE	CKA_CLASS = CKO_SECRET_KEY CKA_PRIVATE = CK_TRUE CKA_KEY_TYPE = CKK_GOST28147
Public key Certificate	C_CreateObject		CKA_PRIVATE = CK_FALSE
Secret key, Private key	C_CreateObject		CKA_PRIVATE = CK_TRUE

A

Дополнительные источники информации

1 PKCS #11 v2.40: Cryptographic Token Interface Standard. OASIS PKCS11 Technical Committee.

2 Расширение PKCS#11 для использования российских криптографических алгоритмов. Москва.

3 RFC4357 — Additional Cryptographic Algorithms for Use with GOST. CRYPTO-PRO.

Указатель

C

A

M

O

Ф