Функции для работы со слотами и токенами

Функции для работы со слотами и токенами

C_GetSlotList()

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

Назначение

Функция позволяет получить список зарегистрированных слотов в системе. Список идентификаторов слотов возвращается в массиве типа **CK_SLOT_ID**. Параметр *tokenPresent* указывает, должен ли список включать слоты только с подключенным токенами (CK_TRUE) или все слоты (CK_FALSE); *pSlotList* указывает на переменную, в которую возвращается количество слотов.

Существует два варианта вызова **C_GetSlotList**:

1. Если *pSlotList* == NULL_PTR, тогда **C_GetSlotList** вернет в **pulCount* только количество слотов, а не полностью список. Содержимое указываемого *pulCount* буфера в таком случае не имеет значения, и функция вернет CKR_OK.
2. Если *pSlotList* != NULL_PTR, тогда **pulCount* должен содержать размер (в формате CK_SLOT_ID элемента) указываемого *pSlotList* буфера. Если размер буфера является достаточным для того, чтобы вместить в себя список, то список возвращается в нем и функция

возвращает CKR_OK. В противном случае **C_GetSlotList** возвращает CKR_BUFFER_TOO_SMALL. В обоих случаях **pulCount* принимает значение, равное количеству слотов.

Поскольку **C_GetSlotList** не выделяет свое пространство, приложение часто может вызывать **C_GetSlotList** дважды (а иногда и больше - если приложение пытается получить лист слотов с подключенными токенами, то количество таких слотов меняется между тем, когда приложение запрашивает количество слотов, и тем, когда приложение запрашивает сами слоты. Тем не менее, несколько вызовов **C_GetSlotList** отнюдь не требуется.

Все слоты, переданные **C_GetSlotList**, должны быть доступны для запросов **C_GetSlotInfo** в качестве корректных слотов. Слоты, доступные через библиотеку PKCS #11, проверяются все время выполнения **C_GetSlotList** для прогнозирования длины списка. Если приложение вызывает **C_GetSlotList** с **не**-NULL значением *pSlotList*, и затем пользователь подключает или отключает устройство, изменившийся список слотов будут видим и корректен лишь в том случае, если функция **C_GetSlotList** будет вызвана снова со значением NULL. Слоты при отключении токенов не удаляются, а помечаются как пустые (tokenPresent == CK_FALSE). Текущее количество слотов при подключении новых токенов может быть увеличено, уменьшение доступно только после вызова **C_Finalize**.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

CKR_ARGUMENTS_BAD,

CKR_BUFFER_TOO_SMALL,

CKR_CRYPTOKI_NOT_INITIALIZED,

CKR_FUNCTION_FAILED,

CKR_GENERAL_ERROR,

CKR_HOST_MEMORY.

Расширенные коды ошибок.

Пример

```
CK_ULONG ulSlotCount, ulSlotWithTokenCount;
CK_SLOT_ID_PTR pSlotList, pSlotWithTokenList;
CK_RV rv;

/*     */
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) {
                /*      */
                .
                .
        }
        free(pSlotList);
}

/*       */
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) {
/*       */
.
.
}
free(pSlotWithTokenList);
```

к содержанию ↑

C_GetSlotInfo

```
CK_DEFINE_FUNCTION(CK_RV, C_GetSlotInfo)(
        CK_SLOT_ID                      slotID,
        CK_SLOT_INFO_PTR        pInfo
);
```

Назначение

Функция позволяет получить информацию о данном слоте. Информация о слоте возвращается в структуре типа **CK_SLOT_INFO**. *slotID* - ID слота, к которому подключен токен, *pInfo* указывает на переменную, которая получает информацию о слоте.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

CKR_ARGUMENTS_BAD,

CKR_CRYPTOKI_NOT_INITIALIZED,

CKR_DEVICE_ERROR,

CKR_FUNCTION_FAILED,

CKR_GENERAL_ERROR,

CKR_HOST_MEMORY,

CKR_SLOT_ID_INVALID.

Расширенные коды ошибок.

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);

        /*      */
        rv = C_GetSlotInfo(pSlotList[0], &slotInfo);
        assert(rv == CKR_OK);

        /*        */
        rv = C_GetTokenInfo(pSlotList[0], &tokenInfo);
        if (rv == CKR_TOKEN_NOT_PRESENT) {
                .
                .
        }
        .
        .
        free(pSlotList);
}
```

к содержанию ↑

C_GetTokenInfo()

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

Назначение

Функция возвращает указатель на структуру **CK_TOKEN_INFO**, в которой хранится информация о токене: модель, метка, название компании-производителя и т.д. *slotID* - ID слота, к которому подключен токен, *pInfo* указывает на место размещения полученной информации о токене.

В связи с особенностью аппаратной реализации у Рутокен нет различия внутренней памяти на открытую и закрытую, поэтому в структуре **CK_TOKEN_INFO** значение полей ulTotalPublicMemory == ulTotalPrivateMemory и ulFreePublicMemory == ulFreePrivateMemory.

При различных условиях могут быть возвращены следующие флаги:

CKF_RNG

CKF_LOGIN_REQUIRED

CKF_USER_PIN_INITIALIZED

CKF_TOKEN_INITIALIZED

CKF_USER_PIN_FINAL_TRY

CKF_USER_PIN_LOCKED

CKF_SO_PIN_COUNT_LOW

CKF_SO_PIN_FINAL_TRY

CKF_SO_PIN_LOCKED

CKF_PROTECTED_AUTHENTICATION_PATH

Следующие флаги не возвращаются никогда:

CKF_WRITE_PROTECTED

CKF_RESTORE_KEY_NOT_NEEDED

CKF_CLOCK_ON_TOKEN

CKF_DUAL_CRYPTO_OPERATIONS

CKF_SECONDARY_AUTHENTICATION

CKF_USER_PIN_COUNT_LOW

CKF_USER_PIN_TO_BE_CHANGED

CKF_SO_PIN_COUNT_LOW

CKF_SO_PIN_TO_BE_CHANGED

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

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_SLOT_ID_INVALID,

CKR_TOKEN_NOT_PRESENT,

CKR_TOKEN_NOT_RECOGNIZED.

Расширенные коды ошибок.

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);

        /*      */
        rv = C_GetSlotInfo(pSlotList[0], &slotInfo);
        assert(rv == CKR_OK);

        /*        */
        rv = C_GetTokenInfo(pSlotList[0], &tokenInfo);
        if (rv == CKR_TOKEN_NOT_PRESENT) {
                .
                .
        }
        .
        .
        free(pSlotList);
}
```

к содержанию ↑

C_WaitForSlotEvent()

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

Назначение

Функция отслеживает события, происходящие со слотами системы (подключение или отключение токена). Параметр *flags* определяет, заблокирован ли вызов **C_WaitForSlotEvent** (т.е. находится ли функция в режиме ожидания события); *pSlot* указывает на переменную типа CK_SLOT_ID, получающую ID слота, с которым произошло событие. Для получения подробной информации о событии необходимо вызвать функцию C_GetSlotInfo. Параметр *pReserved* зарезервирован для следующих версий библиотеки, и для данной версии должен иметь значение NULL_PTR.

На данный момент для использования в *flags о*пределен только один аргумент - **CKF_DONT_BLOCK**.

Каждое приложение, использующее библиотеку PKCS #11, имеет флаг для каждого слота, использующийся для отслеживания нераспознанных событий в слоте. При инициализации библиотеки **C_Initalize** все флаги событий слотов сбрасываются, т.е.независимо от наличия/отсутствия токенов очередь событий в слотах будет пуста.

Если при вызове **C_WaitForSlotEvent** выставлены флаг **CKF_DONT_BLOCK** и флаг наличия события в каком-либо слоте, функция возвращает ID того слота, в котором произошло событие в переменную, на которую указывает *pSlot*. Если во время вызова функции флаги выставлены у нескольких слотов, то библиотекой выбирается, какой флаг будет сброшен и возвращен ID слота. Если **C_WaitForSlotEvent** вызвана из нескольких потоков приложения, то при появлении одного события все ожидающие его потоки завершат ожидание и продолжат работу.

Если при вызове **C_WaitForSlotEvent** выставлен флаг **CKF_DONT_BLOCK** и нет выставленных флагов наличия события ни для одного из слотов, функция возвращает CKR_NO_EVENT и в этом случае содержимое переменной, на которую указывает *pSlot*, не определено.

Если при вызове **C_WaitForSlotEvent** сброшен флаг **CKF_DONT_BLOCK**, то функция выполняется аналогично описанному выше сценарию, за исключением того, что она будет заблокирована. Это означает, что если во время вызова функции нет выставленных флагов событий в слотах, **C_WaitForSlotEvent** будет ждать до тех пор, пока какой-нибудь из флагов не будет выставлен. Если **C_WaitForSlotEvent** находится в режиме ожидания (заблокирована), то вызов **C_Finalize** из другого потока прервет ожидание и завершит работу библиотеки, а **C_WaitForSlotEvent** вернет значение CKR_CRYPTOKI_NOT_INITIALIZED.

В реализации для Рутокен события для слотов происходят при подключении\отключении токена. Для одного слота хранится одно событие, т.к. хранить более одного события не имеет смысла.

События регистрируются все время между вызовами **C_Initalize** и **C_Finalize**, т.е все время пока библиотека готова к использованию. При вызове **C_WaitForSlotEvent** первое событие из очереди изымается, пользователю возвращается ID слота, с которым оно произошло. Очередь становится короче на единицу.

Если требуется получить все события, то **C_WaitForSlotEvent** требуется вызвать столько раз, пока очередной вызов не вернет ошибку CKR_NO_EVENT, это указывает на то, что очередь событий пуста и новых больше не приходило.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

CKR_ARGUMENTS_BAD,

CKR_CRYPTOKI_NOT_INITIALIZED,

CKR_FUNCTION_FAILED,

CKR_GENERAL_ERROR,

CKR_HOST_MEMORY,

CKR_NO_EVENT.

Пример

```
CK_FLAGS flags = 0;
CK_SLOT_ID slotID;
CK_SLOT_INFO slotInfo;

.
.
/*        */
rv = C_WaitForSlotEvent(flags, &slotID, NULL_PTR);
assert(rv == CKR_OK);

/*      */
rv = C_GetSlotInfo(slotID, &slotInfo);
assert(rv == CKR_OK);
.
.
```

к содержанию ↑

C_GetMechanismList()

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

Назначение

Функция позволяет получить список механизмов, поддерживаемых токеном. *SlotID* - ID слота, к которому подключен токен; *pulCount* указывает на переменную типа CK_ULONG, куда возвращается количество механизмов. Список механизмов возвращается в структуре типа CK_MECHANISM_TYPE.

Указатель *pMechanismList* может иметь значение NULL_PTR. В этом случае функция возвращает только количество механизмов (но не список), для которых необходимо зарезервировать память.

Существует два варианта вызова функции **C_GetMechanismList** приложением:

1. Если указатель *pMechanismList* имеет значение NULL_PTR, тогда **C_GetMechanismList** возвращает лишь количество механизмов (но не список). Содержимое указываемого *pMechanismList* буфера в таком случае не имеет смысловой нагрузки, и функция вернет CKR_OK.
2. Если *pMechanismList* != NULL_PTR, тогда **pulCount* должен содержать размер (в элементах CK_MECHANISM_TYPE) указываемого *pMechanismList* буфера. Если размер буфера является достаточным для того, чтобы вместить в себя список, то список возвращается в нем и функция возвращает CKR_OK. В противном случае **C_GetMechanismList** возвращает CKR_BUFFER_TOO_SMALL. В обоих случаях **pMechanismList* принимает значение, равное количеству механизмов.

Различные типы токенов поддерживают разное количество механизмов, более подробная информация доступна в 3.2.6 Механизмы стандарта PKCS#11 и 3.2.7 Механизмы расширения стандарта PKCS#11.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

CKR_ARGUMENTS_BAD,

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_SLOT_ID_INVALID,

CKR_TOKEN_NOT_PRESENT,

CKR_TOKEN_NOT_RECOGNIZED.

Расширенные коды ошибок.

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);
}
```

к содержанию ↑

C_GetMechanismInfo()

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

Назначение

Функция позволяет получить информацию о данном механизме для данного токена. Информация о механизме возвращается в структуре типа **CK_MECHANISM_INFO**. *slotID* - ID слота, к которому подключен токен, *type* - тип механизма, *pInfo* указывает на переменную, которая получает информацию о механизме.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

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_MECHANISM_INVALID,

CKR_SLOT_ID_INVALID,

CKR_TOKEN_NOT_PRESENT,

CKR_TOKEN_NOT_RECOGNIZED.

Расширенные коды ошибок.

Пример

```
CK_SLOT_ID slotID;
CK_MECHANISM_INFO info;
CK_RV rv;

.
.

/*     CKM_MD5    */
rv = C_GetMechanismInfo(slotID, CKM_MD5, &info);
if (rv == CKR_OK) {
        if (info.flags & CKF_DIGEST) {
    .
    .
        }
}
```

к содержанию ↑

C_InitToken()

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

- slotID - ID слота, к которому подключен токен;
- pPin - указатель на начальный PIN-код Администратора (который не обязательньо должен быть нуль-терминированной строкой);
- ulPinLen - длина PIN-кода в байтах;
- pLabel - указатель на метку токена (которая должна быть дополнена пустыми символами (пробелами) до 32 байт и не быть нуль-терминированной строкой).

Данный стандарт позволяет значению PIN-кода содержать любой корректный UTF8 символ, но производители токена могут накладывать свои ограничения (только цифры?).

Назначение

Функция инициализирует память токена, а именно:

- сбрасывает счетчик неверных попыток ввода пин-кода пользователя,
- устанавливает пин-код по умолчанию для пользователя,
- очищает папку для объектов PKCS#11,
- устанавливает метку токена.

Если токен не был инициализирован (т.е. является новым, с производства), тогда параметр *pPin* становится первоначальным значением PIN-кода Администратора. Если токен инициализируется повторно, то *pPin* сверяется с текущим PIN-кодом Администратора для авторизации операции инициализации. В обоих случаях при успешном выполнении функции PIN-код Администратора принимает значение параметра *pPin*. Ес ли PIN-код Администратора утерян, то токен может быть инициализирован повторно с использованием расширенного производителем механизма C_EX_InitToken. Флаг **CKF_TOKEN_INITIALIZED** в структуре **CK_TOKEN_INFO** показывает действие, которое станет результатом вызова **C_InitToken**. Если он установлен, то токен будет инициализирован повторно, и пользователь должен будет предоставить текущий пароль Администратора.

Когда токен инициализируется, все объекты, которые могут быть уничтожены, будут уничтожены (т.е. все объекты, за исключением "неуничтожаемых", таких как ключи, сгенерированные на самом токене). Также будет недоступен доступ обычного пользователя до тех пор, пока Администратор не установит PIN-код пользователя.

Токен не может быть инициализирован, если библиотека обнаружила, что какое-либо приложение имеет открытую сессию с ним; когда вызов **C_InitToken** осуществляется при таких обстоятельствах, то он завершается ошибкой **CKR_SESSION_EXISTS**. К сожалению, может случиться так, что при вызове **C_InitToken** некоторые другие приложения имеют открытую сессию с токеном, но библиотека не может этого обнаружить, поскольку она не может знать все о других приложениях, использующих токен. В этом случае последствия вызова **C_InitToken** не определены.

При форматировании устройств Рутокен из Rutoken Control Panel можно задать политику смены PIN-кода пользователя – «PIN-код пользователя может менять только пользователь». Стандарт PKCS#11 не предполагает такой политики смены PIN-кода пользователя, поэтому при попытке установки PIN-кода по-умолчанию для пользователя будет возвращена ошибка CKR_FUNCTION_REJECTED. Для смены политики смены PIN-кода следует использовать функцию **C_EX_InitToken** из расширения стандарта PKCS #11.

Обработка входных параметров функции происходит следующим образом.

1. Если *ulPinLen* != 0 и *pPin* != NULL_PTR, то проверяется длина PIN-кода. Если длина не является допустимой, то возвращается значение **CKR_ARGUMENTS_BAD**. Если длина удовлетворяет условиям проверки, то проверяются права доступа для «CKU_SO» на основании предъявленных *pPin* и *ulPinLen.*
2. Если *ulPinLen* != 0 и *pPin* == NULL_PTR, то возвращается ошибка **CKR_ARGUMENTS_BAD**.
3. Если *ulPinLen* == 0 и *pPin* != NULL_PTR, то возвращается ошибка **CKR_ARGUMENTS_BAD.**
4. Если *pLabel* == NULL_PTR и *pPin* != NULL_PTR, то возвращается ошибка **CKR_ARGUMENTS_BAD**.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

CKR_ARGUMENTS_BAD,

CKR_CRYPTOKI_NOT_INITIALIZED,

CKR_DEVICE_ERROR,

CKR_DEVICE_MEMORY,

CKR_DEVICE_REMOVED,

CKR_FUNCTION_CANCELED,

CKR_FUNCTION_FAILED,

CKR_FUNCTION_REJECTED,

CKR_GENERAL_ERROR,

CKR_HOST_MEMORY,

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.

Расширенные коды ошибок.

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) {
  .
  .
}
```

к содержанию ↑

C_InitPIN()

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

- *hSession* - дескриптор сессии;
- *pPin* - указатель на PIN-код Пользователя;
- *ulPinLen* - длина PIN-кода в байтах.

Данный стандарт позволяет значению PIN-кода содержать любой корректный UTF8 символ, но производители токена могут накладывать свои ограничения (только цифры?).

Назначение

Функция производит инициализацию PIN-кода Пользователя токена.

Функция может быть вызвана только в сесии «R/W SO Functions». При попытке вызвать функцию в других сессиях будет возвращена ошибка CKR_USER_NOT_LOGGED_IN.

Функция обрабатывает входные параметры следующим образом.

1. Если *ulPinLen* != 0 и *pPin* != NULL_PTR, то проверяется длина PIN-кода. Если длина не является допустимой, то возвращается значение **CKR_PIN_LEN_RANGE**. Если длина удовлетворяет условиям проверки, то проверяются права доступа для «CKU_USER» на основании предъявленных *pPin* и *ulPinLen*.
2. Если *ulPinLen* != 0 и *pPin* == NULL_PTR, то возвращается ошибка **CKR_ARGUMENTS_BAD**.
3. Если *ulPinLen* == 0 и *pPin* != NULL_PTR, то возвращается ошибка **CKR_PIN_LEN_RANGE.**
4. Если *ulPinLen* == 0 и *pPin* == NULL_PTR, то возвращается ошибка **CKR_ARGUMENTS_BAD.**

При форматировании устройств Рутокен из Rutoken Control Panel можно задать политику смены PIN-кода пользователя – «PIN-код пользователя может менять только пользователь». Стандарт PKCS#11 не предполагает такой политики смены PIN-кода пользователя, поэтому при попытке инициализации PIN-кода будет возвращена ошибка CKR_DEVICE_ERROR. Для инициализации токена с возможность задать политику смены PIN-кода следует использовать функцию **C_EX_InitToken** из расширения стандарта PKCS #11.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

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_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.

Расширенные коды ошибок.

Пример

```
CK_SESSION_HANDLE hSession;
CK_UTF8CHAR newPin[]= {"NewPIN"};
CK_RV rv;

rv = C_InitPIN(hSession, newPin, sizeof(newPin)-1);
if (rv == CKR_OK) {
        .
        .
        }
```

к содержанию ↑

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
);
```

- *hSession* - дескриптор сессии;
- *pOldPin* - указатель на старый PIN-код Пользователя;
- *ulOldLen* - длина старого PIN-кода в байтах;
- *pNewPin* - указатель на новый PIN-код Пользователя;
- *ulNewLen* - длина нового PIN-кода в байтах.

Данный стандарт позволяет значению PIN-кода содержать любой корректный UTF8 символ, но производители токена могут накладывать свои ограничения (только цифры?).

Назначение

Функция изменяет PIN-код пользователя, выполнившего авторизацию на токене.

Функция может быть вызвана только в сессиях «R/W Public Session», «R/W SO Functions» или «R/W User Functions». При попытке вызвать функцию в других сессиях будет возвращена ошибка CKR_SESSION_READ_ONLY.

Функция обрабатывает входные параметры следующим образом.

1. Если *ulPinLen* != 0 и *pPin* != NULL_PTR, то проверяется длина PIN-кода. Если длина не является допустимой, то возвращается значение **CKR_PIN_LEN_RANGE**. Если длина удовлетворяет условиям проверки, то проверяются права доступа для «CKU_USER» на основании предъявленных *pPin* и *ulPinLen.*
2. Если *ulPinLen* != 0 и *pPin* == NULL_PTR, то возвращается ошибка **CKR_ARGUMENTS_BAD**.
3. Если *ulPinLen* == 0 и *pPin* != NULL_PTR, то возвращается ошибка **CKR_PIN_LEN_RANGE.**
4. Если *ulPinLen* == 0 и *pPin* == NULL_PTR, то возвращается ошибка **CKR_ARGUMENTS_BAD).**

При форматировании устройств Рутокен из Rutoken Control Panel можно задать политику смены PIN-кода пользователя – «PIN-код пользователя может менять только пользователь». Стандарт PKCS#11 не предполагает такой политики смены PIN-кода пользователя, поэтому при попытке смены PIN-кода будет возвращена ошибка CKR_DEVICE_ERROR. Для инициализации токена с возможность задать политику смены PIN-кода следует использовать функцию **C_EX_InitToken** из расширения стандарта PKCS #11.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Стандартные коды ошибок:

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_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.

Расширенные коды ошибок.

Пример

```
CK_SESSION_HANDLE hSession;
CK_UTF8CHAR oldPin[] = {"OldPIN"};
CK_UTF8CHAR newPin[] = {"NewPIN"};
CK_RV rv;

rv = C_SetPIN(hSession, oldPin, sizeof(oldPin), newPin, sizeof(newPin));
if (rv == CKR_OK) {
  .
  .
}
```

к содержанию ↑