

Описание функций расширения

C_EX_TokenManage

- 1 Функции общего назначения
 - 1.1 C_EX_GetFunctionListExtended()
- 2 Функции для работы со слотами и токенами
 - 2.1 C_EX_GetTokenInfoExtended()
 - 2.2 C_EX_InitToken()
 - 2.3 C_EX_UnblockUserPIN()
 - 2.4 C_EX_SetTokenName()
 - 2.5 C_EX_GetTokenName()
 - 2.6 C_EX_SetLicense()
 - 2.7 C_EX_GetLicense()
 - 2.8 C_EX_SetLocalPIN()
 - 2.9 C_EX_TokenManage()
 - 2.10 C_EX_SlotManage
- 3 Функции для работы с сертификатами
 - 3.1 C_EX_GetCertificateInfoText()
 - 3.2 C_EX_CreateCSR()
- 4 Функции для работы с CMS/PKCS#7 сообщениями
 - 4.1 C_EX_PKCS7Sign()
 - 4.2 C_EX_PKCS7VerifyInit()
 - 4.3 C_EX_PKCS7Verify()
 - 4.4 C_EX_PKCS7VerifyUpdate()
 - 4.5 C_EX_PKCS7VerifyFinal()
- 5 Функции для работы с флеш-памятью
 - 5.1 C_EX_GetDriveSize()
 - 5.2 C_EX_FormatDrive()
 - 5.3 C_EX_GetVolumesInfo()
 - 5.4 C_EX_ChangeVolumeAttributes()
- 6 Функции для работы с журналом
 - 6.1 C_EX_GetJournal()
- 7 Функции для работы с беспроводным каналом связи
 - 7.1 C_EX_LoadActivationKey()
 - 7.2 C_EX_SetActivationPassword()
 - 7.3 C_EX_GenerateActivationPassword()
 - 7.4 C_EX_TokenManage()
- 8 Вспомогательные функции
 - 8.1 C_EX_FreeBuffer()

Функции общего назначения

C_EX_GetFunctionListExtended()

Назначение

Функция возвращает список дополнительных функций, расширяющих библиотеку PKCS#11 для работы с Рутокен.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_GetFunctionListExtended)(
    CK_FUNCTION_LIST_EXTENDED_PTR_PTR    ppFunctionList
);
```

Параметры

<i>ppFunctionList</i>	[out]	указатель на список функций расширения
-----------------------	-------	--

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Пример

```
HMODULE hModule; // PKCS #11
CK_FUNCTION_LIST_EXTENDED_PTR pFunctionListEx; // CK_FUNCTION_LIST_EXTENDED,
CK_C_EX_GetFunctionListExtended pfGetFunctionListEx; // C_EX_GetFunctionListExtended
CK_RV rv; //

hModule = LoadLibrary("rtPKCS11.dll"); //
pfGetFunctionListEx = (CK_C_EX_GetFunctionListExtended)
    GetProcAddress(hModule, "C_EX_GetFunctionListExtended"); // C_EX_GetFunctionListExtended
if (pfGetFunctionListEx == NULL_PTR) //
    printf ("Loading library -> Failed \n");
else
    printf ("Loading library -> OK \n");

rv = pfGetFunctionListEx(&pFunctionListEx); //
if (rv == CKR_OK) //
    printf ("Getting extended function list -> OK \n");
else
    printf ("Getting extended function list -> Failed \n");
.
.
FreeLibrary(hModule); //
```

[к содержанию ↑](#)

Функции для работы со слотами и токенами

C_EX_GetTokenInfoExtended()

Назначение

Функция позволяет получить специфическую для устройств Рутокен информацию: класс токена, количество свободной и общей памяти, тип токена, номер протокола, количество оставшихся попыток ввода пин-кода Пользователя и Администратора и т.д. По сравнению с похожей по назначению стандартной функции **C_GetTokenInfo** функция расширения предоставляет более полную информацию о токене.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_GetTokenInfoExtended)(
    CK_SLOT_ID          slotID,
    CK_TOKEN_INFO_EXTENDED_PTR pInfo
);
```

Параметры

<i>slotID</i>	[in]	ID слота, к которому подключен токен
<i>pInfo</i>	[out]	указатель на структуру CK_TOKEN_INFO_EXTENDED, получающую расширенные данные токена

Возвращаемые значения

CKR_OK – функция выполнена успешно.

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.

Пример

```
CK_SLOT_ID          slotID;                                //
CK_TOKEN_INFO_EXTENDED ckTokenInfoEx;                      //
CK_RV               rv;                                    //

.
.
ckTokenInfoEx.ulSizeofThisStructure = sizeof(CK_TOKEN_INFO_EXTENDED); //
CK_TOKEN_INFO_EXTENDED
rv = pfGetFunctionListEx -> C_EX_GetTokenInfoExtended(slotID, &ckTokenInfoEx); //
if (rv ==
CKR_OK)
//
printf ("Getting Extended Token Info -> OK \n");
else
printf ("Getting Extended Token Info -> Failed \n");
```

[к содержанию ↑](#)

C_EX_InitToken()

Назначение

Функция позволяет полностью инициализировать память токена. На этапе инициализации есть возможность задать политику смены PIN-кода пользователя, метки токена произвольной длины и т.д.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_InitToken)(
    CK_SLOT_ID          slotID,
    CK_UTF8CHAR_PTR     pPin,
    CK_ULONG            ulPinLen,
    CK_RUTOKEN_INIT_PARAM_PTR pInitInfo
);
```

Параметры

slotID	[in]	ID слота, к которому подключен токен для инициализации
pPin	[in]	указатель на PIN-код Администратора
ulPinLen	[in]	длина PIN-кода Администратора, в байтах

<i>pInitInfo</i>	[in]	указатель на структуру CK_RUTOKEN_INIT_PARAM, хранящую параметры инициализации
------------------	------	--

Примечание

Основное отличие функции **C_EX_InitToken** от похожей по назначению стандартной функции **C_InitToken** в том, что функция расширения полностью инициализирует память токена, а также предоставляет возможность задать политику смены PIN-кода пользователя, метку произвольной длины, а стандартная функция **C_InitToken** позволяет инициализировать только память, занятую объектами PKCS#11, и задать метку фиксированной длины.

Все параметры, необходимые для инициализации, передаются в структуре типа **CK_RUTOKEN_INIT_PARAM**.

Внимание! Для **Рутокен S** параметры ulMinAdminPinLen и ulMinUserPinLen из структуры **CK_RUTOKEN_INIT_PARAM**, не могут принимать значение больше 1.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

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_LOCKED,

CKR_SESSION_EXISTS,

CKR_SLOT_ID_INVALID,

CKR_TOKEN_NOT_PRESENT.

Пример

```

static CK_CHAR USER_PIN[] = {'1', '2', '3', '4', '5', '6', '7', '8'};          // PIN-
static CK_CHAR SO_PIN[] = {'8', '7', '6', '5', '4', '3', '2', '1'};          // PIN-
CK_FUNCTION_LIST_PTR
pFunctionList;                                                                //
CK_FUNCTION_LIST,
CK_C_EX_GetFunctionList pfGetFunctionList;                                  //
C_GetFunctionList
CK_SLOT_ID slots
[100];
//
CK_ULONG ulSlotCount =
100;                                                                //
CK_RUTOKEN_INIT_PARAM ckRtInitParams;                                     //
,
CK_RV
rv;
//

/* CK_RUTOKEN_INIT_PARAM */
ckRtInitParams.ulSizeofThisStructure = sizeof(CK_RUTOKEN_INIT_PARAM);      //
ckRtInitParams.UseRepairMode =
0;                                                                // PIN- (0 - , , !0 - ,
)
ckRtInitParams.pNewAdminPin = SO_PIN;                                     //
    PIN- ( (?), 32 )
ckRtInitParams.ulNewAdminPinLen = sizeof(SO_PIN);                      // PIN-
ckRtInitParams.pNewUserPin = USER_PIN;
// PIN- ( (?), 32 )
ckRtInitParams.ulNewUserPinLen = sizeof (USER_PIN);                    // PIN-
ckRtInitParams.ChangeUserPINPolicy = TOKEN_FLAGS_ADMIN_CHANGE_USER_PIN
| TOKEN_FLAGS_USER_CHANGE_USER_PIN; //
    PIN- :
ckRtInitParams.ulMinAdminPinLen = 6;                                     //
    PIN- ( 6, 32 )
ckRtInitParams.ulMinUserPinLen =
6;                                                                // PIN- ( 6, 32 )
ckRtInitParams.ulMaxAdminRetryCount = 10;                                //
    PIN- ( 3, 10)
ckRtInitParams.ulMaxUserRetryCount = 10;                                //
    PIN- ( 1, 10)
ckRtInitParams.pTokenLabel = "Rutoken label";                          //
ckRtInitParams.ulLabelLen =
13;                                                                //
.
.
pfGetFunctionList = (CK_C_GetFunctionList)GetProcAddress(hModule, "C_GetFunctionList"); //
pfGetFunctionList(&pFunctionList);

rv = pfGetFunctionList -> C_GetSlotList(CK_TRUE, slots, &ulSlotCount);    //
if (rv == CKR_OK
)
//
    printf("Getting connected slots list -> OK \n");
else
    printf("Getting connected slots list -> failed \n");
if (ulSlotCount == 0)
    printf("No Rutoken is available");

rv = pfGetFunctionListEx ->
    C_EX_InitToken(slots[0], SO_PIN, sizeof(SO_PIN), &ckRtInitParams);    // (, )
if (rv ==
CKR_OK)
//
    printf("Token initialization -> OK \n");
else
    printf("Token initialization -> failed \n");

```

[к содержанию ↑](#)

C_EX_UnblockUserPIN()

Назначение

Функция позволяет разблокировать PIN-код Пользователя.

Синтаксис

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

Параметры

<i>hSession</i>	[in]	дескриптор сессии
-----------------	------	-------------------

Примечание

Функция сбрасывает счетчик неверных попыток ввода PIN-кода для CKU_USER. Функция может быть вызвана только из сессии, имеющей статус CKS_RW_SO_FUNCTIONS.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

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_USER_NOT_LOGGED_IN,

CKR_ARGUMENTS_BAD.

Пример

```

static CK_CHAR SO_PIN[] = {'8', '7', '6', '5', '4', '3', '2', '1'};           // PIN-
CK_SESSION_HANDLE
hSession;                                                                    //
CK_SLOT_ID slots
[100];
//
CK_ULONG ulSlotCount =
100;                                                                           //
CK_RV
rv;
//
.
.
rv = pfGetFunctionList -> C_OpenSession(                                     //
    slots[0],                                                                //
    CKF_SERIAL_SESSION | CKF_RW_SESSION),                                   // /
    0,                                                                       //
    0,                                                                       //
    &hSession);                                                             //
if (rv != CKR_OK)                                                            //
    printf("Opening session -> failed \n");
else
    printf("Opening session -> OK \n");

rv = pfGetFunctionList -> C_Login(                                           // ,
    hSession,                                                                //
    CKU_SO,                                                                  // ,
    SO_PIN,                                                                  // PIN-
    sizeof(SO_PIN));                                                         // PIN-
if (rv != CKR_OK)                                                            //
    printf("Logging -> failed \n");
else
    printf("Logging -> OK \n");

rv = pfGetFunctionListEx -> C_EX_UnblockUserPIN(hSession);                  // PIN-
if (rv != CKR_OK)                                                            //
    printf("Unblocking UserPIN -> failed \n");
else
    printf("Unblocking UserPIN -> OK \n");
.
.
pfGetFunctionList -> C_Logout(hSession);                                     //
pfGetFunctionList -> C_CloseSession(hSession);                             //

```

[к содержанию ↑](#)

C_EX_SetTokenName()

Назначение

Функция позволяет задать метку токена (символьное имя) произвольной длины.

Синтаксис

```

CK_DEFINE_FUNCTION(CK_RV, C_EX_SetTokenName)(
    CK_SESSION_HANDLE    hSession,
    CK_BYTE_PTR          pLabel,
    CK_ULONG              ulLabelLen
);

```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>pLabel</i>	[in]	указатель на буфер, содержащий новую метку токена
<i>uiLabelLen</i>	[in]	размер буфера с новой меткой токена, в байтах

Примечание

Функция позволяет установить метку токена произвольной длины, в отличие от стандартной функции **C_InitToken**, которая позволяет устанавливать метку токена фиксированного размера. Функция может быть вызвана только из сессии, имеющей статус CKS_RW_USER_FUNCTIONS.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

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_PIN_EXPIRED,

CKR_SESSION_CLOSED,

CKR_SESSION_HANDLE_INVALID,

CKR_SESSION_READ_ONLY,

CKR_USER_NOT_LOGGED_IN.

Пример

```

CK_SESSION_HANDLE hSession;                                //
CK_CHAR ckcNewLabel[100];                                // 100
CK_RV rv;                                                  //

.
.
pfGetFunctionList -> C_OpenSession(slots[0], CKF_SERIAL_SESSION | CKF_RW_SESSION), 0, 0, &hSession); //
pfGetFunctionList -> C_Login(hSession, CKU_USER, USER_PIN, sizeof
(USER_PIN));                                              //

strcpy ((char *)ckcNewLabel, "Label is set by C_EX_SetTokenName from rtPKCS11ecp.
dll");                                                  //
rv = pfGetFunctionList -> C_EX_SetTokenName(hSession,
ckcNewLabel, 32);                                        //
if (rv !=
CKR_OK)
//
    printf("Changing of label -> failed \n");
else
    printf("Changing of label -> OK \n");
.
.
pfGetFunctionList -> C_Logout
(hSession);
//
pfGetFunctionList -> C_CloseSession
(hSession);
//

```

[к содержанию ↑](#)

C_EX_GetTokenName()

Назначение

Функция позволяет получить метку токена произвольной длины. Функция может быть вызвана из любой сессии.

Синтаксис

```

CK_DEFINE_FUNCTION(CK_RV, C_EX_GetTokenName)(
    CK_SESSION_HANDLE    hSession,
    CK_BYTE_PTR          pLabel,
    CK_ULONG_PTR         ulLabelLen
);

```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>pLabel</i>	[out]	указатель на буфер, содержащий метку токена
<i>ulLabelLen</i>	[in, out]	размер буфера с меткой токена, в байтах

Примечание

Функция позволяет получить метку токена произвольной длины, в отличие от стандартной функции **C_GetTokenInfo**, которая позволяет получить метку токена фиксированного размера.

Для определения необходимого количества памяти для хранения метки токена, необходимо вызвать функцию **C_EX_GetTokenName** с параметром *pLabel* равным NULL_PTR, тогда в параметре *pulLabelLen* будет возвращен указатель на переменную с требуемым размером буфера.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

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_SESSION_CLOSED,

CKR_SESSION_HANDLE_INVALID,

CKR_BUFFER_TOO_SMALL.

Пример

```
CK_BYTE ckLabel[1000];
CK_ULONG ckulLabelLen = 0;

.
.
rv = pfGetFunctionListEx -> C_EX_GetTokenName(hSession, ckLabel, &ckulLabelLen);
//
if (rv !=
CKR_OK)
//
    printf("Getting name of token -> failed \n");
else
    printf("Getting name of token -> OK \n");
```

[к содержанию ↑](#)

C_EX_SetLicense()

Назначение

Функция предназначена для записи данных вендора в токен и позволяет записать лицензию на токен.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_SetLicense)(
    CK_SESSION_HANDLE    hSession,
    CK_ULONG              ulLicenseNum,
    CK_BYTE_PTR           pLicense,
    CK_ULONG              ulLicenseLen
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
-----------------	------	-------------------

<i>ulLicenseNum</i>	[in]	идентификатор лицензии. Параметр может принимать одно из двух значений: 1 или 2
<i>pLicense</i>	[in]	указатель на буфер, содержащий лицензию
<i>ulLicenseLen</i>	[in]	размер буфера с лицензией, в байтах. Параметр может принимать только одно значение: 72

Возвращаемые значения

CKR_OK – функция выполнена успешно.

CKR_ARGUMENTS_BAD,

CKR_CRYPTOKI_NOT_INITIALIZED,

CKR_DEVICE_ERROR,

CKR_DEVICE_MEMORY,

CKR_DEVICE_REMOVED,

CKR_FUNCTION_FAILED,

CKR_FUNCTION_NOT_SUPPORTED,

CKR_GENERAL_ERROR,

CKR_HOST_MEMORY,

CKR_PIN_EXPIRED,

CKR_SESSION_CLOSED,

CKR_SESSION_HANDLE_INVALID,

CKR_SESSION_READ_ONLY,

CKR_USER_NOT_LOGGED_IN.

Примечание

Формат данных (лицензии) определяет вендор, размер ограничивается 72 байтами. Всего может быть до двух лицензий. Функция может быть вызвана только из сессии, имеющей статус CKS_RW_USER_FUNCTIONS или CKS_RW_SO_FUNCTIONS.

При инициализации памяти токена с помощью функций **C_InitToken** или **C_EX_InitToken** ранее записанная лицензия удалена не будет.

Пример

Пример использования функции C_EX_SetLicense

```
CK_ULONG      ckulLicenseNumber = 1;                                     //
CK_BYTE      ckbLicense[100];
//
CK_ULONG      ckulLicenseLen = 0;                                       //

.
.
pfGetFunctionList -> C_OpenSession(slots[0], (CKF_SERIAL_SESSION), 0, 0,&hSession); //
pfGetFunctionList -> C_Login(hSession, CKU_SO, SO_PIN, sizeof(SO_PIN));
//

pfGetFunctionListEx ->
    C_EX_GetLicense(hSession, ckulLicenseNumber, ckbLicense, &ckulLicenseLen); //

for (CK_ULONG ckulIndex=0; ckulIndex < ckulLicenseLen; ckulIndex++) // :
    ckbLicense[ckulIndex] = (72 - ckulIndex);

rv = pfLstEx->C_EX_SetLicense(hSession, ckulLicenseNumber, ckbLicense, ckulLicenseLen); //
if (rv != CKR_OK)
    printf("C_EX_SetLicense() -> failed\n");
else
    printf("C_EX_SetLicense() -> OK\n");
.
.
pfGetFunctionList -> C_Logout(hSession); //
pfGetFunctionList -> C_CloseSession(hSession); //
```

[к содержанию ↑](#)

C_EX_GetLicense()

Назначение

Функция позволяет прочитать лицензию, записанную на токен.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_GetLicense)(
    CK_SESSION_HANDLE    hSession,
    CK_ULONG              ulLicenseNum,
    CK_BYTE_PTR           pLicense,
    CK_ULONG_PTR          pulLicenseLen
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>ulLicenseNum</i>	[in]	идентификатор лицензии. Параметр может принимать одно из двух значений: 1 или 2
<i>pLicense</i>	[out]	указатель на буфер для получения лицензии
<i>ulLicenseLen</i>	[in, out]	размер буфера с лицензией, в байтах. Параметр может принимать только одно значение: 72

Возвращаемые значения

CKR_OK – функция выполнена успешно.

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_SESSION_CLOSED,
CKR_SESSION_HANDLE_INVALID,
CKR_FUNCTION_NOT_SUPPORTED.

Примечание

Функция позволяет прочитать лицензию с токена. Функция может быть вызвана из сессии любого состояния. Длина лицензии может иметь единственное значение 72 байта, либо при передаче *pLicense* равного NULL_PTR в параметре *pullLicenseLen* возвращается указатель на длину буфера для требуемой лицензии.

Пример

Пример использования функции C_EX_SetLicense

```
CK_ULONG      ckulLicenseNumber = 1;                                //
CK_BYTE        ckbLicense[100];
//
CK_ULONG      ckulLicenseLen = 0;                                    //

.
.
pfGetFunctionList -> C_OpenSession(slots[0], (CKF_SERIAL_SESSION), 0, 0,&hSession);      //

rv = pfGetFunctionListEx -> C_EX_GetLicense(hSession, ckulLicenseNumber, ckbLicense, &ckulLicenseLen);
//
if (rv != CKR_OK)
    printf("C_EX_GetLicense() -> failed\n");
else
    printf("C_EX_GetLicense() -> OK\n");
.
.
pfGetFunctionList -> C_CloseSession(hSession);                        //
```

[к содержанию ↑](#)

C_EX_SetLocalPIN()

Назначение

Функция устанавливает локальный PIN-код, если он не был установлен или меняет локальный PIN-код, если он был установлен заранее.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_SetLocalPIN)(
    CK_SLOT_ID          slotID,
    CK_UTF8CHAR_PTR     pUserPin,      // pOldLocalPin
    CK_ULONG             ulUserPinLen,  // pOldLocalPinLen
    CK_UTF8CHAR_PTR     pNewLocalPin,
    CK_ULONG             ulNewLocalPinLen,
    CK_ULONG             ulLocalID
);
```

Параметры

slotID	[in]	идентификатор слота, к которому подключен токен
pUserPin или pOldLocalPin	[in]	указатель на текущий PIN-код Пользователя или на текущий локальный PIN-код
ulUserPinLen или pOldLocalPinLen	[in]	длина текущего PIN-кода Пользователя или длина текущего локального PIN-кода
pNewLocalPin	[in]	указатель на новый Локальный PIN-код
ulNewLocalPinLen	[in]	длина нового Локального PIN-кода
ulLocalID	[in]	идентификатор Локального PIN-кода в пределах от 0x03 до 0x1E

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Пример

```
CK_BYTE Pin[] = {'1', '2', '3', '4', '5', '6', '7', '8'};           // PIN- PIN-
CK_SLOT_ID slots[100];
//
.
.
rv = pfGetFunctionListEx -> C_EX_SetLocalPIN(
    slots
[0],                                                                //
,
Pin,
// PIN- PIN-
    arraysize
(Pin),                                                            // PIN-
    PIN-
        "00000000000000000000000000000000",                    // PIN-
30,
// PIN-
0x1F
// PIN-
);
if (rv !=
CKR_OK)                                                            //
    printf("C_EX_SetLocalPIN() -> failed \n");
else
    printf("C_EX_SetLocalPIN() -> OK \n");
```

[к содержанию ↑](#)

C_EX-TokenManage()

Назначение

Предоставляет механизм принудительной смены PIN-кода пользователя.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX-TokenManage)(
    CK_SESSION_HANDLE    hSession,
    CK_ULONG              ulMode,
    CK_VOID_PTR           pValue
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>ulMode</i>	[in]	режим работы функции: MODE_RESET_CUSTOM_PIN_TO_STANDARD – сбросить кастомизированный PIN-код по-умолчанию на стандартный. MODE_RESET_PIN_TO_DEFAULT – сброс PIN-кода на PIN-код по-умолчанию MODE_CHANGE_DEFAULT_PIN – установить кастомизированный PIN-код по-умолчанию. MODE_FORCE_USER_TO_CHANGE_PIN – установить флаг принудительной смены PIN-кода.
<i>pValue</i>	[in]	- при режиме MODE_RESET_CUSTOM_PIN_TO_STANDARD в качестве параметра pValue передается указатель на тип пользователя (CKU_USER или CKU_SO), для которого сбрасывается PIN-код по-умолчанию - при режиме MODE_RESET_PIN_TO_DEFAULT в качестве параметра pValue передается указатель на тип пользователя (CKU_USER или CKU_SO или локальные пользователи), для которого сбрасывается PIN-код - при режиме MODE_CHANGE_DEFAULT_PIN в качестве параметра pValue передается указатель на структуру CK_VENDOR_PIN_PARAMS: typedef struct CK_VENDOR_PIN_PARAMS { CK_USER_TYPE userType; CK_UTF8CHAR_PTR pPinValue; CK_ULONG ulPinLength; } CK_VENDOR_PIN_PARAMS; userType – тип пользователя, для которого устанавливаем кастомизированный PIN-код (CKU_USER или CKU_SO) pPinValue – массив байт содержащий кастомизированный PIN-код по-умолчанию - при режиме MODE_FORCE_USER_TO_CHANGE_PIN в качестве параметра pValue передается указатель на тип пользователя (CKU_USER или CKU_SO), которого мы хотим заставить сменить PIN-код:

Возвращаемые значения

CKR_OK – функция выполнена успешно.

CKR_ARGUMENTS_BAD – неправильно переданы параметры функции

CKR_USER_NOT_LOGGED_IN – пользователь SKU_SO не авторизован на токене (режимы)

Мультифункциональная функция

 Здесь описаны не все возможные параметры функции. Описание других параметров функции можно найти [здесь](#)

[к содержанию ↑](#)

C_EX_SlotManage

Назначение

Позволяет:

1. получить расширенную информацию о всех локальных PIN-кодах токена.

2. получить информацию об установленном флаге принудительной смене PIN-кода

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_SlotManage) (  
    CK_SLOT_ID slotID,  
    CK_ULONG ulMode,  
    CK_VOID_PTR pValue  
)
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>ulMode</i>	[in]	режим работы функции: MODE_GET_LOCAL_PIN_INFO – получение информации о локальных PIN-кодах MODE_GET_PIN_SET_TO_BE_CHANGED – получение информации об принудительной смене PIN-кода
<i>pValue</i>	[out]	- При режиме MODE_GET_LOCAL_PIN_INFO – указатель, на структуру CK_LOCAL_PIN_INFO: typedef struct CK_LOCAL_PIN_INFO { CK_ULONG ulPinID; CK_ULONG ulMinSize; CK_ULONG ulMaxSize; CK_ULONG ulMaxRetryCount; CK_ULONG ulCurrentRetryCount; CK_FLAGS flags; } CK_LOCAL_PIN_INFO; ulPinID – идентификатор локального PIN-кода (входной параметр). - При режиме MODE_GET_PIN_SET_TO_BE_CHANGED – указатель тип пользователя (CKU_USER или CKU_SO), для которого хотим узнать установлен ли флаг принудительной смены PIN-кода.

Для режима работы функции MODE_GET_LOCAL_PIN_INFO в *pValue* возвращается указатель массив структур типа CK_LOCAL_PIN_INFO:

<i>ulPinID</i>	[in]	идентификатор PIN-кода
<i>ulMinSize</i>	[out]	заданная минимальная длина PIN-кода
<i>ulMaxSize</i>	[out]	заданная максимальная длина PIN-кода
<i>ulMaxRetryCount</i>	[out]	заданное максимальное значение счетчика неверных попыток ввода пароля
<i>ulCurrentRetryCount</i>	[out]	текущее значение счетчика оставшихся попыток ввода пароля: 0 – PIN-код заблокирован
<i>flags</i>	[out]	информационные флаги: LOCAL_PIN_FLAGS_NOT_DEFAULT – локальный PIN-код не является PIN-кодом по умолчанию LOCAL_PIN_FLAGS_FROM_SCREEN – PIN-код может введен только с экрана

Возвращаемые значения

- CKR_OK – функция выполнена успешно.
- CKR_PIN_EXPIRED – если PIN-код нужно сменить смене (MODE_GET_PIN_SET_TO_BE_CHANGED).
- CKR_ARGUMENTS_BAD – неверно указаны аргументы функции.

[к содержанию ↑](#)

Функции для работы с сертификатами

C_EX_GetCertificateInfoText()

Назначение

Функция позволяет получить информацию о сертификате из токена в текстовом виде.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_GetCertificateInfoText)(
    CK_SESSION_HANDLE      hSession,
    CK_OBJECT_HANDLE       hCert,
    CK_CHAR_PTR*           pInfo,
    CK_ULONG_PTR           pulInfoLen
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>hCert</i>	[in]	дескриптор объекта (сертификата)
<i>pInfo</i>	[out]	указатель на буфер, содержащий текстовую информацию о сертификате
<i>pulInfoLen</i>	[out]	размер буфера, в байтах

Примечание

Функция может быть вызвана из любого состояния. Так как память для массива выделяется внутри функции, по окончании работы функции *pInfo* нужно высвободить функцией **C_EX_FreeBuffer**.

[к содержанию ↑](#)

C_EX_CreateCSR()

Назначение

Создает запрос на выпуск сертификата в центр сертификации и упаковывает его в формат PKCS#10.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_CreateCSR)(
    CK_SESSION_HANDLE      hSession,
    CK_OBJECT_HANDLE       hPublicKey,
    CK_CHAR_PTR            *dn,
    CK_ULONG               dnLength,
    CK_BYTE_PTR            *pCsr,
    CK_ULONG_PTR           pulCsrLength,
    CK_OBJECT_HANDLE       hPrivKey,
    CK_CHAR_PTR            *pAttributes,
    CK_ULONG               ulAttributesLength,
    CK_CHAR_PTR            *pExtensions,
    CK_ULONG               ulExtensionsLength
);
```

Параметры

<i>hSession</i>	[i n]	дескриптор сессии
<i>hPublicKey</i>	[i n]	дескриптор открытого ключа для создания сертификата
<i>dn</i>	[i n]	уникальное имя – массив строк, где в первой строке располагается тип поля в текстовом формате или идентификатор объекта (CN или «2.5.4.3»). Во второй строке должно находиться значение поля в кодировке UTF8. Последующие поля передаются аналогично, количество строк должно быть кратно двум
<i>dnLength</i>	[i n]	количество строк в массиве, на который ссылается параметр <i>dn</i>
<i>pCsr</i>	[i n]	указатель на указатель на массив байт, в который будет записан созданный запрос на сертификат
<i>pulCsrLength</i>	[i n]	указатель на переменную, в которой сохраняется размер массива.
<i>hPrivKey</i>	[i n]	закрытый ключ, соответствующий открытому ключу, на который ссылается параметр <i>hPublicKey</i> . Если значение установлено в «0», то поиск закрытого ключа будет осуществляться по ID открытого ключа.
<i>pAttributes</i>	[i n]	дополнительные атрибуты для включения в запрос. Формат аналогичен формату параметра <i>dn</i> (например, «1.2.840.113549.1.9.7» или «challengePassword»).
<i>ulAttributeLength</i>	[i n]	количество атрибутов в массиве, на который указывает параметр <i>attributes</i> .
<i>pExtensions</i>	[i n]	расширения для включения в запрос. Формат аналогичен параметру <i>dn</i>
<i>ulExtensionsLength</i>	[i n]	количество расширений в параметрах <i>extensions</i>

Примечание

Функция может быть вызвана только из состояний "R/W User Functions" и "R User Functions". Память для массива *pCsr* выделяется внутри функции, поэтому после окончания работы память необходимо освободить, вызвав функцию **C_EX_FreeBuffer()**.

Функция поддерживает генерацию запроса на сертификат для следующих типов ключей: ГОСТ 34.10-2012 с длиной закрытого ключа 256 бит, ГОСТ 34.10-2001 и RSA (типы ключей СКК_GOSTR3410 и СКК_RSA).

Возвращаемые значения

СКР_OK – функция выполнена успешно.

Пример

[к содержанию ↑](#)

Функции для работы с CMS/PKCS#7 сообщениями

C_EX_PKCS7Sign()

Назначение

Подписывает переданные на вход данные в формате PKCS#7.

Синтаксис

```

CK_DEFINE_FUNCTION(CK_RV, C_EX_PKCS7Sign)(
    CK_SESSION_HANDLE      hSession,
    CK_BYTE_PTR            pData,
    CK_ULONG                ulDataLen,
    CK_OBJECT_HANDLE        hCert,
    CK_BYTE_PTR            *ppEnvelope,
    CK_ULONG_PTR            pEnvelopeLen,
    CK_OBJECT_HANDLE        hPrivKey,
    CK_OBJECT_HANDLE_PTR    phCertificates,
    CK_ULONG                ulCertificatesLen,
    CK_ULONG                flags
);

```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>pData</i>	[in]	указатель на байт-массив CK_BYTE, содержащий данные для подписи
<i>ulDataLen</i>	[in]	длина данных для подписи
<i>hCert</i>	[in]	дескриптор сертификата
<i>ppEnvelope</i>	[out]	указатель на байт-массив, в который передаются данные (сообщение)
<i>pEnvelopeLen</i>	[out]	указатель на длину созданного буфера с сообщением
<i>hPrivKey</i>	[in]	дескриптор закрытого ключа, соответствующего указанному сертификату. Если равен 0, то закрытый ключ будет искаться по ID сертификата
<i>phCertificates</i>	[in]	указатель на массив сертификатов, цепочку сертификатов
<i>ulCertificatesLen</i>	[in]	количество сертификатов в параметре <i>phCertificates</i>
<i>flags</i>	[in]	переменная типа CK_ULONG, может принимать следующие значения: 0 – хеширование программное, исходные данные, на которые ссылается указатель <i>pData</i> , сохраняются вместе с подписанным сообщением; PKCS7_DETACHED_SIGNATURE – исходные данные, на которые ссылается указатель <i>pData</i> , не сохраняются вместе с подписанным сообщением. USE_HARDWARE_HASH - использовать аппаратное хеширование.

Примечание

Функция может быть вызвана только из состояний "R/W User Functions" и "R User Functions". Буфер *ppEnvelope* создается внутри функции. После окончания работы с ним необходимо освободить его, вызвав функцию **C_EX_FreeBuffer()**.

Функция поддерживает подпись по следующим алгоритмам: ГОСТ 34.10-2012 с длиной закрытого ключа 256 бит и ГОСТ 34.10-2001 (механизм СКМ_GOSTR3410).

Возвращаемые значения

CKR_OK – функция выполнена успешно.

[к содержанию ↑](#)

C_EX_PKCS7VerifyInit()

Назначение

Инициализирует процесс проверки подписи переданных на вход данных в формате PKCS#7.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_PKCS7VerifyInit)(
    CK_SESSION_HANDLE      hSession,
    CK_BYTE_PTR            pCms,
    CK_ULONG                ulCmsSize,
    CK_VENDOR_X509_STORE_PTR pStore,
    CK_VENDOR_CRL_MODE      ckMode,
    CK_FLAGS                flags
);

typedef struct CK_VENDOR_X509_STORE {
    CK_VENDOR_BUFFER_PTR    pTrustedCertificates;           //
    CK_ULONG                ulTrustedCertificateCount;      //
    CK_VENDOR_BUFFER_PTR    pCertificates;                 // ,
    CK_ULONG                ulCertificateCount;              //
    CK_VENDOR_BUFFER_PTR    pCrls;                         //
    CK_ULONG                ulCrlCount;                     //
} CK_VENDOR_X509_STORE;

typedef struct CK_VENDOR_BUFFER {
    CK_BYTE_PTR pData;           //
    CK_ULONG ulSize;             //
}
```

Параметры

<i>hSession</i>	[i n]	дескриптор сессии
<i>pCms</i>	[i n]	указатель на массив байтов, содержащий сообщение в формате PKCS#7 для проверки
<i>ulCmsSize</i>	[i n]	длина сообщения
<i>pStore</i>	[i n]	указатель на структуру типа CK_VENDOR_X509_STORE, которая содержит указатели на необходимые для проверки подписи доверенные сертификаты, сертификаты подписывающей стороны и списки отозванных сертификатов
<i>ckMode</i>	[i n]	политика проверки списков отозванных сертификатов (CRLs), может принимать следующие значения: OPTIONAL_CRL_CHECK – отсутствие списка отозванных сертификатов не вызывает ошибку при проверке подписи; LEAF_CRL_CHECK – требуется наличие списка отозванных сертификатов для проверки сертификата подписанта; ALL_CRL_CHECK – требуется наличие списка отозванных сертификатов для проверки каждого сертификата из цепочки
<i>flags</i>	[i n]	опции проверки подписи, могут принимать следующие значения: CKF_VENDOR_DO_NOT_USE_INTERNAL_CMS_CERTS – не использовать содержащиеся в CMS сообщении сертификаты для поиска сертификатов подписантов (сертификаты должны быть заданы в структуре CK_VENDOR_X509_STORE); CKF_VENDOR_ALLOW_PARTIAL_CHAINS – для проверки подписи достаточно, чтобы хотя бы один сертификат удостоверяющего центра из цепочки сертификатов подписывающей стороны находился в списке доверенных.

Примечание

Функция может быть вызвана только из состояний "R/W User Functions" и "R User Functions".

Возвращаемые значения

CKR_OK – функция выполнена успешно,
CKR_ARGUMENTS_BAD,

CKR_OPERATION_ACTIVE,

CKR_FUNCTION_FAILED,
CKR_FUNCTION_NOT_SUPPORTED.

[к содержанию ↑](#)

C_EX_PKCS7Verify()

Назначение

Выполняет проверку присоединенной (attached) подписи в формате PKCS#7 и возвращает данные, которые были подписаны, и сертификаты подписантов.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_PKCS7Verify)(
    CK_SESSION_HANDLE          hSession,
    CK_BYTE_PTR_PTR            ppData,
    CK_ULONG_PTR                pulDataSize,
    CK_VENDOR_BUFFER_PTR_PTR   ppSignerCertificates,
    CK_ULONG_PTR                pulSignerCertificatesCount
);

typedef struct CK_VENDOR_BUFFER {
    CK_BYTE_PTR pData;           //
    CK_ULONG ulSize;             //
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>ppData</i>	[out]	указатель на массив байтов, содержащий данные, которые были подписаны (EncapsulatedContentInfo)
<i>pulDataSize</i>	[out]	размер данных
<i>ppSignerCertificates</i>	[out]	указатель на массив, содержащий сертификаты подписантов
<i>pulSignerCertificatesCount</i>	[out]	количество сертификатов в массиве

Примечание

Функция может быть вызвана только из состояний "R/W User Functions" и "R User Functions", после вызова функции **C_EX_PKCS7Verifyinit**.

Возвращаемые значения

CKR_OK – функция выполнена успешно.
CKR_CERT_CHAIN_NOT_VERIFIED – функция выполнена успешно, но цепочка сертификации не проверялась,
CKR_ARGUMENTS_BAD,
CKR_CERT_CHAIN_NOT_VERIFIED,
CKR_FUNCTION_NOT_SUPPORTED,
CKR_HOST_MEMORY,
CKR_OPERATION_NOT_INITIALIZED,
CKR_SIGNATURE_INVALID.

[к содержанию ↑](#)

C_EX_PKCS7VerifyUpdate()

Назначение

Начинает процесс проверки отсоединенной (detached) подписи в формате PKCS#7.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_PKCS7VerifyUpdate)(
    CK_SESSION_HANDLE          hSession,
    CK_BYTE_PTR                pData,
    CK_ULONG                    ulDataSize
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>pData</i>	[in]	указатель на массив байтов, содержащий блок данных, которые были подписаны (EncapsulatedContentInfo)
<i>ulDataSize</i>	[in]	размер данных

Примечание

Функция может быть вызвана только из состояний "R/W User Functions" и "R User Functions", после вызова функции **C_EX_PKCS7VerifyInit**. Для завершения процесса проверки подписи необходим вызов функции **C_EX_PKCS7VerifyFinal**.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

CKR_ARGUMENTS_BAD,

CKR_FUNCTION_NOT_SUPPORTED,

CKR_HOST_MEMORY,

CKR_OPERATION_NOT_INITIALIZED.

[к содержанию ↑](#)

C_EX_PKCS7VerifyFinal()

Назначение

Завершает процесс проверки отсоединенной (detached) подписи в формате PKCS#7 и возвращает сертификаты подписантов.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_PKCS7VerifyFinal)(
    CK_SESSION_HANDLE          hSession,
    CK_VENDOR_BUFFER_PTR_PTR  ppSignerCertificates,
    CK_ULONG_PTR                pulSignerCertificatesCount
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>ppSignerCertificates</i>	[out]	указатель на массив, содержащий сертификаты подписантов

<i>pulSignerCertificatesCount</i>	[out]	количество сертификатов в массиве
-----------------------------------	-------	-----------------------------------

Примечание

Функция может быть вызвана только из состояний "R/W User Functions" и "R User Functions", после вызова функции **C_EX_PKCS7VerifyUpdate**.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

CKR_SIGNATURE_INVALID,

CKR_ARGUMENTS_BAD,

CKR_CERT_CHAIN_NOT_VERIFIED,

CKR_FUNCTION_NOT_SUPPORTED,

CKR_HOST_MEMORY,

CKR_OPERATION_NOT_INITIALIZED.

[к содержанию](#) ↑

Функции для работы с флеш-памятью

C_EX_GetDriveSize()

Назначение

Возвращает размер флеш-памяти токена.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_GetDriveSize)(
    CK_SLOT_ID          slotID,
    CK_ULONG_PTR        pulDriveSize
);
```

Параметры

<i>slotID</i>	[in]	идентификатор слота с подключенным токеном
<i>pulDriveSize</i>	[out]	возвращаемый размер флеш-памяти в Мб

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Пример

```
CK_ULONG ulDriveSize = 0; // -

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

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

[к содержанию ↑](#)

C_EX_FormatDrive()

Разбивает флеш-память токена на независимые разделы с разными правами доступа.

Функция приводит к переподключению токена

После окончания работы этой функции происходит переподключение токена. Нет никаких гарантий, что работу с этим токеном можно производить по старому SLOT_ID: он может начать указывать на другой токен или не быть валидным вовсе. Новый SLOT_ID токена можно найти, например, по серийному номеру.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_FormatDrive) (
    CK_SLOT_ID          slotID,
    CK_USER_TYPE         userType,
    CK_UTF8CHAR_PTR     pPin,
    CK_ULONG             ulPinLen,
    CK_VOLUME_FORMAT_INFO_EXTENDED_PTR pInitParams,
    CK_ULONG             ulInitParamsCount
);

typedef struct CK_VOLUME_FORMAT_INFO_EXTENDED
{
    CK_ULONG             ulVolumeSize;
    CK_ACCESS_MODE_EXTENDED accessMode;
    CK_OWNER_EXTENDED    volumeOwner;
    CK_FLAGS             flags;
} CK_VOLUME_FORMAT_INFO_EXTENDED;
```

Параметры

<i>slotID</i>	[i n]	идентификатор слота с подключенным токеном
<i>userType</i>	[i n]	тип пользователя, допустимое значение только CKU_SO – глобальный администратор.
<i>pPin</i>	[i n]	PIN-код пользователя
<i>ulPinLen</i>	[i n]	длина PIN-кода пользователя

<i>plnitParams</i>	[i n]	указатель на массив структур типа CK_VOLUME_FORMAT_INFO_EXTENDED, содержащих детальную информацию о разделах	
		<i>ulVolumeSize</i>	размер раздела в Мб
		<i>accessMode</i>	флаги доступа к разделу: ACCESS_MODE_RW – доступ на чтение и запись; ACCESS_MODE_RO – доступ только на чтение; ACCESS_MODE_HIDDEN – скрытый раздел, защищенный от отображения в операционной системе, чтения, записи и любого другого типа доступа; ACCESS_MODE_CD – раздел, эмулирующий CD-ROM.
		<i>volumeOwner</i>	владелец раздела, имеет права на изменение флага доступа к разделу: CKU_USER – глобальный пользователь; CKU_SO – глобальный администратор; CKU_LOCAL_X, где X – число от 1 до 8 – локальный пользователь X
		<i>flags</i>	остальные флаги
<i>ullnitParamsCount</i>	[i n]	количество записей в массиве	

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Пример

```

CK_ULONG          VolumeRWSize = 0;          //
CK_ULONG          VolumeROSize = 0;          //
CK_ULONG          VolumeCDSIZE = 0;          //    CD-ROM
CK_ULONG          VolumeHISize = 0;          //

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

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

...

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

printf("\nFormatting flash memory");
rv = pFunctionListEx->C_EX_FormatDrive( aSlots[0],          //

CKU_SO,          //

SO_PIN,          //    PIN-

(SO_PIN),          //    PIN-          sizeof

InitParams,          //          sizeof(InitParams)); //

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

```

[к содержанию ↑](#)

C_EX_GetVolumesInfo()

Назначение

Возвращает информацию о существующих на флеш-памяти разделах.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_GetVolumesInfo)(
    CK_SLOT_ID          slotID,
    CK_VOLUME_INFO_EXTENDED_PTR pInfo,
    CK_ULONG_PTR         pulInfoCount
);

typedef struct CK_VOLUME_INFO_EXTENDED
{
    CK_VOLUME_ID_EXTENDED idVolume;
    CK_ULONG              ulVolumeSize;
    CK_ACCESS_MODE_EXTENDED accessMode;
    CK_OWNER_EXTENDED     volumeOwner;
    CK_FLAGS              flags;
} CK_VOLUME_INFO_EXTENDED;
```

Параметры

slotID	[in]	идентификатор слота с подключенным токеном
pInfo	[out]	указатель на массива структур типа CK_VOLUME_INFO_EXTENDED с информацией о разделах:
	idVolume	идентификатор раздела, может принимать значения от 1 до 9
	ulVolumeSize	размер раздела в Мб (не больше общего объема носителя)
	accessMode	флаги доступа к разделу: ACCESS_MODE_RW – доступ на чтение и запись; ACCESS_MODE_RO – доступ только на чтение; ACCESS_MODE_HIDDEN – скрытый раздел, защищенный от отображения в операционной системе, чтения, записи и любого другого типа доступа; ACCESS_MODE_CD – раздел, эмулирующий CD-ROM.
	volumeOwner	владелец раздела, имеет права на изменение флага доступа к разделу: CKU_USER – глобальный пользователь; CKU_SO – глобальный администратор; CKU_LOCAL_X, где X – число от 1 до 8 – локальный пользователь X
	flags	остальные флаги
pullInfoCount	[out]	размер массива

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Примечания

При вызове функции с пустым указателем *pInfo* функция возвращает размер массива в *pulInfoCount*.

Пример

```

CK_VOLUME_INFO_EXTENDED_PTR  pVolumesInfo = NULL_PTR; //
CK_ULONG                      ulVolumesInfoount = 0;    //

printf("\nGetting volumes info");
rv = pFunctionListEx->C_EX_GetVolumesInfo( aSlots[0],           //
NULL_PTR,                                     //
                                           &ulVolumesInfoount); //

pVolumesInfo = (CK_VOLUME_INFO_EXTENDED*)malloc(ulVolumesInfoount * sizeof(CK_VOLUME_INFO_EXTENDED));
memset(pVolumesInfo,
      0,
      (ulVolumesInfoount * sizeof(CK_VOLUME_INFO_EXTENDED)));

rv = pFunctionListEx->C_EX_GetVolumesInfo(aSlots[0],           //
pVolumesInfo,                                     //
                                           &ulVolumesInfoount); //
if (rv != CKR_OK)
{
    printf(" -> Failed\n");
}
else
{
    printf(" -> OK\n");
    for (i = 0; i < (int)ulVolumesInfoount; i++)
    {
        printf("\nPrinting volume %1.2d info:\n", (int)i+1);
        printf(" Volume id:           %2.2d \n", pVolumesInfo[i].idVolume);           //
        printf(" Volume size:         %d Mb \n", pVolumesInfo[i].ulVolumeSize);       //
        printf(" Access mode:         %2.2d \n", pVolumesInfo[i].accessMode);         //
        printf(" Volume owner:        %2.2d \n", pVolumesInfo[i].volumeOwner);       //
        printf(" Flags:               0x%8.8X \n", pVolumesInfo[i].flags);          //
    }
}

```

[к содержанию ↑](#)

C_EX_ChangeVolumeAttributes()

Назначение

Функция изменяет флаг доступа к разделу.

Функция приводит к переоподключению токена



После изменения доступа к разделу на постоянной основе происходит переоподключение токена. Нет никаких гарантий, что работу с этим токеном можно производить по старому SLOT_ID: он может начать указывать на другой токен или не быть валидным вовсе. Новый SLOT_ID токена можно найти, например, по серийному номеру.

Синтаксис

```

CK_DEFINE_FUNCTION(CK_RV, C_EX_ChangeVolumeAttributes)(
    CK_SLOT_ID          slotID,
    CK_USER_TYPE        userType,
    CK_UTF8CHAR_PTR     pPin,
    CK_ULONG            ulPinLen,
    CK_VOLUME_ID_EXTENDED idVolume,
    CK_ACCESS_MODE_EXTENDED newAccessMode,
    CK_BBOOL            bPermanent
);

```

Параметры

<i>slotID</i>	[i n]	идентификатор слота с подключенным токеном
<i>userType</i>	[i n]	владелец раздела, допустимые значения: CKU_USER – глобальный пользователь; CKU_SO – глобальный администратор; число X от 3 до 8 – локальный пользователь X
<i>pPin</i>	[i n]	PIN-код пользователя
<i>ulPinLen</i>	[i n]	длина PIN-кода пользователя
<i>idVolume</i>	[i n]	идентификатор раздела
<i>newAccess Mode</i>	[i n]	новые флаги доступа к разделу
<i>bPermanent</i>	[i n]	флаг режима изменения атрибута: CK_TRUE – постоянное изменение атрибута доступа, действует вплоть до следующего изменения атрибутов; CK_FALSE – временное изменение прав доступа, действует до первого отключения устройства из USB-порта, после чего права доступа сбрасываются на прежние.

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Пример

```
printf("\nChanging volume attributes");
rv = pFunctionListEx->C_EX_ChangeVolumeAttributes(aSlots[0],          //
                                                    CKU_SO,             //
                                                    SO_PIN,             // PIN-
                                                    sizeof(SO_PIN),      // PIN-
                                                    VolumeR0,          //
                                                    ACCESS_MODE_RW,      //
                                                    CK_TRUE);            // CK_TRUE -   , CK_FALSE -

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

[к содержанию ↑](#)

Функции для работы с журналом

C_EX_GetJournal()

Назначение

Функция возвращает журнал операций.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_GetJournal)(
    CK_SLOT_ID          slotID,
    CK_BYTE_PTR          pJournal,
    CK_ULONG_PTR         pulJournalSize
);
```

Параметры

<i>slotID</i>	[in]	идентификатор слота
<i>pJournal</i>	[out]	указатель на запись журнала
<i>pulJournalSize</i>	[out]	размер записи журнала

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Примечания

Журнал позволяет хранить только одну запись с информацией о последней выполненной операции подписи. Неудачные попытки подписи в журнале не фиксируются. Описание формата записи журнала доступно [по ссылке](#).

Для получения записи журнала необходимо вызвать функцию C_EX_GetJournal() с переданными в нее значением слота, к которому подключен Рутокен ЭЦП, и данными буфера, в который будет возвращена запись журнала. Вызов функции C_EX_GetJournal() с указателем на NULL вместо буфера вернет длину записи журнала.

Перед запросом информации о журнале необходимо выполнить аутентификацию Пользователем.

Получение значения журнала

```

CK_BYTE_PTR      pJournal = NULL_PTR;          //
CK_ULONG         ulJournalSize = 0;            //

while(TRUE)
{
    ...
    /*      */
    printf("Getting journal size");
    rv = pFunctionListEx->C_EX_GetJournal(aSlots[0],      //
                                           NULL_PTR,      //
                                           &ulJournalSize);//

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

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

    /*      */
    printf("Getting journal");
    rv = pFunctionListEx->C_EX_GetJournal(aSlots[0],      //
                                           pJournal,      //
                                           &ulJournalSize);//

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

    ...
    break;
}

```

[к содержанию ↑](#)

Функции для работы с беспроводным каналом связи

Secure Messaging

C_EX_LoadActivationKey()

Назначение

Активирует защищенный канал связи с токеном с использованием пароля (поддерживается только в 20-й версии прошивки)

Синтаксис

```

CK_DEFINE_FUNCTION(CK_RV, C_EX_LoadActivationKey)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR      key,
    CK_ULONG         keySize
);

```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>key</i>	[in]	указатель на пароль
<i>keySize</i>	[in]	длина пароля

Возвращаемые значения

CKR_OK – функция выполнена успешно.

[к содержанию ↑](#)

C_EX_SetActivationPassword()

Назначение

Активирует защищенный канал связи с токеном с использованием пароля (поддерживается с 21-й версии прошивки).

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_SetActivationPassword)(
    CK_SLOT_ID      slotID,
    CK_UTF8CHAR_PTR password
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>password</i>	[in]	пароль активации

Возвращаемые значения

CKR_OK – функция выполнена успешно.

[к содержанию ↑](#)

C_EX_GenerateActivationPassword()

Назначение

Генерирует пароль для активации защищенного канала связи с токеном.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_GenerateActivationPassword)(
    CK_SESSION_HANDLE hSession,
    CK_ULONG          ulPasswordNumber,
    CK_UTF8CHAR_PTR   pPassword,
    CK_ULONG_PTR       pulPasswordSize,
    CK_ULONG           ulPasswordCharacterSet
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>ulPasswordNumber</i>		порядковый номер генерируемого пароля. Допустимые значения от 1 до 6 и GENERATE_NEXT_PASSWORD – генерировать следующий пароль (только для 21-й версии микропрограммы)
<i>pPassword</i>	[out]	указатель на буфер, содержащий сгенерированный пароль
<i>pulPasswordSize</i>	[out]	размер буфера
<i>ulPasswordCharacterSet</i>	[in]	набор допустимых символов: CAPS_AND_DIGITS – заглавные буквы латинского алфавита без О и цифры без 0 CAPS_ONLY – заглавные буквы латинского алфавита

Возвращаемые значения

CKR_OK – функция выполнена успешно.

[к содержанию ↑](#)

C_EX_TokenManage()

Назначение

Управляет режимом работы токена и таймаутом беспроводного соединения Рутокен Bluetooth.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_TokenManage)(
    CK_SESSION_HANDLE    hSession,
    CK_ULONG              ulMode,
    CK_VOID_PTR           pValue
);
```

Параметры

<i>hSession</i>	[in]	дескриптор сессии
<i>ulMode</i>	[in]	режим работы функции: MODE_SET_BLUETOOTH_POWEROFF_TIMEOUT – установить таймаут для беспроводного канала связи MODE_SET_CHANNEL_TYPE – установить тип используемого канала связи
<i>pValue</i>	[in]	- значение таймаута для режима MODE_SET_BLUETOOTH_POWEROFF_TIMEOUT: 1 .. 0x46 – произвольное значение в минутах, от 1 минуты до 70 минут BLUETOOTH_POWEROFF_TIMEOUT_DEFAULT – по умолчанию BLUETOOTH_POWEROFF_TIMEOUT_MAX – максимальное значение (70 минут) - канал связи для режима MODE_SET_CHANNEL_TYPE: CHANNEL_TYPE_USB – по шине USB CHANNEL_TYPE_BLUETOOTH – по Bluetooth

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Мультифункциональная функция

 Здесь описаны не все возможные параметры функции. Описание других параметров функции можно найти [здесь](#)

[к содержанию ↑](#)

Вспомогательные функции

C_EX_FreeBuffer()

Назначение

Функция освобождает память, выделенную другими расширенными функциями, например **C_EX_GetCertificateInfoText**.

Синтаксис

```
CK_DEFINE_FUNCTION(CK_RV, C_EX_FreeBuffer)(  
    CK_BYTE_PTR          pBuffer  
);
```

Параметры

<i>pBuffer</i>	[in]	указатель на буфер
----------------	------	--------------------

Возвращаемые значения

CKR_OK – функция выполнена успешно.

Пример

```
CK_BYTE_PTR          pInfo  
  
.  
.  
rv = pfGetFunctionListEx -> C_EX_FreeBuffer(pInfo);           // ,  
C_EX_GetCertificateInfoText()  
if (rv !=  
CKR_OK)                                                         //  
    printf("C_EX_FreeBuffer() -> failed \n");  
else  
    printf("C_EX_FreeBuffer() -> OK \n");
```

[к содержанию ↑](#)