

ЗАТВЕРДЖЕНО

Наказ Адміністрації Державної служби
спеціального зв'язку та захисту
інформації України

_____.2011 № _____

ТЕХНІЧНІ СПЕЦИФІКАЦІЇ

інтерфейсів засобів криптографічного захисту інформації, що реалізують
алгоритми ДСТУ ГОСТ 28147:2009, ГОСТ 34.311-95,
ДСТУ 4145-2002 згідно з PKCS#11

1. Загальні положення

1.1. Ці технічні специфікації (надалі - Технічні специфікації) визначають інтерфейс прикладного програмування (API, Application Programming Interface), що називається «Cryptoki», для засобів криптографічного захисту інформації, які зберігають криптографічну інформацію та виконують криптографічні функції. Cryptoki, що є скороченням від «інтерфейс криптографічного токєну» (cryptographic token interface), слідує об'єктно-орієнтованому підходу, відповідає цілі незалежності від технології (від будь-якого типу пристрою) та відповідає цілі спільного використання/ розподілу ресурсів (доступ декількох прикладних програм до декількох пристроїв), надаючи прикладним програмам загальний пристрій під назвою «криптографічний токєн».

1.2. Технічні специфікації є обов'язковими для апаратних засобів криптографічного захисту інформації таких, як смарт-карти, РСМСІА-карти, інші апаратні засоби реалізації криптографічних операцій, що забезпечують надійне/ безпечне зберігання закритої/ секретної ключової компоненти під контролем одного користувача. З таким пристроєм криптографічні програми самі не виконують криптографічні операції, а використовують пристрій для виконання операцій із забезпеченням конфіденційності такої інформації як особисті/ секретні ключі, які ніколи не розкриваються/не залишають пристрій. Технічні специфікації є рекомендованими для програмних пристроїв криптографічного захисту інформації.

1.3. Технічні специфікації визначають типи даних, об'єкти, механізми та функції, доступні програмам, що вимагають криптографічних послуг, з використанням мови програмування ANSI C. Ці типи даних і функції надаються постачальником бібліотеки Cryptoki, як правило, через заголовки C. Реалізація бібліотеки Cryptoki може бути виконана будь-якою мовою програмування.

1.4. Засіб криптографічного захисту інформації, що реалізує інтерфейс прикладного програмування, визначений цими Технічними специфікаціями, повинен у своїй експлуатаційній документації містити опис реалізованих (не

реалізованих) типів даних, об'єктів та механізмів, визначених цими Технічними специфікаціями, а також даних, об'єктів та механізмів, визначених виробником, у разі їх наявності. Реалізація інших типів даних, об'єктів та механізмів, не визначених цими Технічними специфікаціями, повинна відповідати вимогам стандарту PKCS#11 версії 2.20.

1.5. Ці Технічні Специфікації визначають вимоги до реалізації криптографічних операцій, що підтримують національні крипто алгоритми ДСТУ ГОСТ 28147:2009, ГОСТ 34.311-95, ДСТУ 4145-2002 відповідно до стандартів ДСТУ ГОСТ 28147:2009, ГОСТ 34.311-95, ДСТУ 4145-2002 та міжнародного стандарту PKCS#11. Вимоги до реалізації інших крипто алгоритмів повинні відповідати зазначеним у стандарті PKCS#11 версії 2.20.

1.6. У цих Технічних специфікаціях терміни вживаються у такому значенні:
ключ шифрування ключа (КШК) – симетричний ключ, на якому здійснюється шифрування симетричного ключа сеансу.

Інші терміни вживаються у значенні, наведеному у Законі України “Про електронний цифровий підпис”, Порядку акредитації центру сертифікації ключів, затвердженому постановою Кабінету Міністрів України від 13.07.2004 № 903, Технічних специфікаціях форматів представлення базових об'єктів, затверджених Наказом Департаменту спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України, Державного департаменту з питань зв'язку та інформатизації Міністерства транспорту та зв'язку України № 99/166 від 11.09.2006, Технічних специфікаціях форматів криптографічних повідомлень, затверджених Наказом Адміністрації Державної служби спеціального зв'язку та захисту інформації України № 112 від 14.05.2010, зі змінами згідно Наказу Адміністрації Державної служби спеціального зв'язку та захисту інформації України № 152 від 15.06.2010, Інструкції про порядок постачання і використання ключів до засобів криптографічного захисту інформації, затвердженої Наказом Адміністрації Державної служби спеціального зв'язку та захисту інформації України № 114 від 12.06.2007, інших нормативно-правових актах з питань криптографічного та технічного захисту інформації.

1.5. Технічні специфікації засновані на міжнародних стандартах:

PKCS#11 “Public Key Cryptography Standard #11 v2.20: Cryptographic Token Interface Standard” (надалі - PKCS#11);

національних стандартах України:

ДСТУ 4145-2002 “Інформаційні технології. Криптографічний захист інформації. Цифровий підпис, що ґрунтується на еліптичних кривих. Формування та перевіряння” (надалі - ДСТУ 4145-2002);

ДСТУ ГОСТ 28147:2009 “Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования” (надалі - ДСТУ ГОСТ 28147:2009),

міждержавних стандартах:

ГОСТ 34.311-95 “Информационная технология. Криптографическая защита информации. Функция хеширования” (надалі - ГОСТ 34.311-95),

нормативно-правових актах:

Інструкція про порядок постачання і використання ключів до засобів криптографічного захисту інформації (затверджена Наказом Адміністрації Державної служби спеціального зв'язку та захисту інформації України №114 від 12.06.2007) (надалі - Наказ №114).

1.6. Якщо в Технічних специфікаціях існують розходження із нормативно-правовими актами та нормативними документами, зазначеними у пункті 1.5 Технічних специфікацій, то використовуються положення цих Технічних специфікацій.

2. Загальна модель та типи даних

2.1. Загальна модель Cryptoki

Загальна модель Cryptoki показана на рис.2.1. Модель починається з одного або декількох застосувань/ програм (Application), які через інші рівні безпеки (Other Security Levels) повинні виконувати певні криптографічні операції, і закінчується одним або декількома криптографічними пристроями (токенами), на яких усі, або деякі з операцій, насправді виконуються. Користувач може бути, або може не бути пов'язаний із застосуванням.

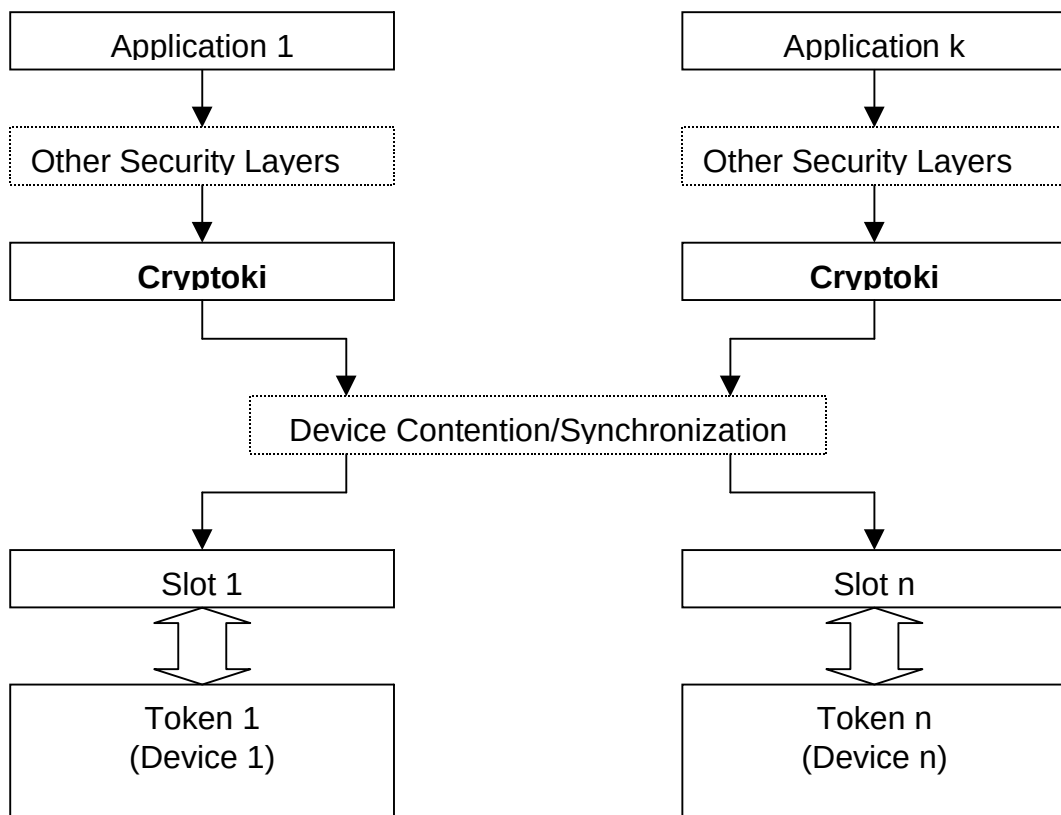


Рис. 2.1. Загальна модель Cryptoki

Cryptoki надає інтерфейс до одного або декількох криптографічних пристроїв (Device), що працюють в системі, через кілька «слотів» (Slot). Кожен слот, що відповідає фізичному зчитувачу (рідеру) чи іншому інтерфейсному пристрою, може містити маркер. Маркер, як правило, «присутній у слоті», коли криптографічні пристрої присутні у рідері. Звичайно, так як Cryptoki забезпечує логічне уявлення слотів і токенів, можуть бути й інші фізичні інтерпретації. Цілком можливо, що кілька слотів можуть мати одні й ті ж фізичні рідери.

Криптографічний пристрій може виконувати деякі криптографічні операції після певного набору команд. Ці команди, як правило, проходить через стандартні драйвери пристроїв, наприклад PCMCIA-карту або службу Socket Service. Cryptoki забезпечує те, що кожен криптографічний пристрій виглядає логічно як і будь-який інший пристрій, незалежно від технології реалізації.

Таким чином, застосування не повинні безпосередньо взаємодіяти з драйверами пристроїв (і навіть не знають, які з них беруть участь); Cryptoki приховує ці деталі.

«Пристрій», що лежать в основі, може бути реалізований повністю на програмному забезпеченні (наприклад, як процес, запущений на сервері), якому не потрібно спеціального устаткування.

Cryptoki реалізується у вигляді інтерфейсної бібліотеки підтримки визначених функцій. Види пристроїв і функцій, що підтримуються, залежать від конкретної реалізації бібліотеки Cryptoki.

Ці Технічні специфікації визначають тільки інтерфейс до бібліотеки, а не їх можливості. Зокрема, не всі бібліотеки будуть підтримувати всі механізми (алгоритми), визначені в цьому інтерфейсі, як і не всі токени/ пристрої будуть підтримувати всі механізми.

2.2. Логічна модель токenu

Логічним поданням Cryptoki токenu є пристрій, який зберігає об'єкти (Object) і може виконувати криптографічні функції.

Cryptoki визначає три класи об'єктів (рис 2.2):

Об'єкт «Дані» (Data) визначається застосуванням.

Об'єкт «Сертифікат» (Certificate) зберігає сертифікат X.509.

Об'єкт «Ключ» (Key) зберігає криптографічний ключ.

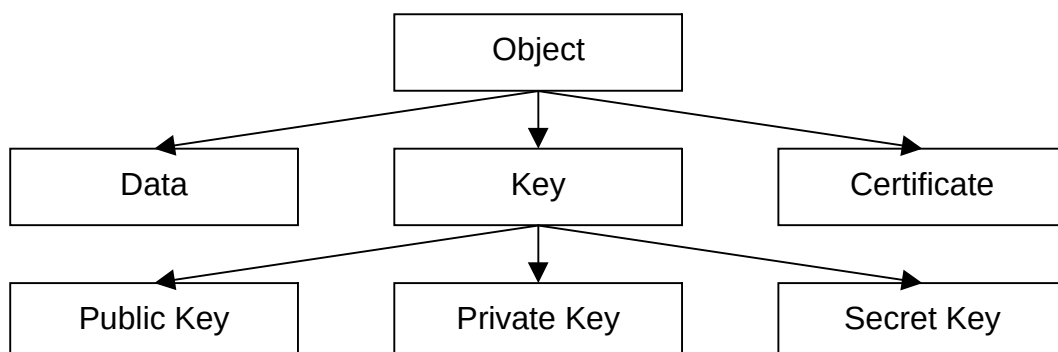


Рис.2.2. Об'єкти токenu

Ключ може бути відкритий ключ (Public Key), особистий/ закритий ключ (Private Key), або секретний ключ (Secret Key). Кожен з цих типів ключів має підтипи для використання в певних механізмах.

Об'єкти, також класифікуються залежно від їх життєвого циклу і видимості.

«Об'єкти токена» (Token objects) є видимими для всіх застосувань, що підключені до токена, та які мають достатньо повноважень, і залишаються на токені навіть після закриття «сесії» (session, з'єднання між застосуванням і токеном) та видалення токена із свого гнізда.

«Сесійні об'єкти» мають тимчасовий характер: всякий раз, коли сесія закривається будь-яким чином, усі сесійні об'єкти, створені під час цієї сесії, автоматично знищуються. Крім того, сесійні об'єкти є видимими тільки застосуванню, яке їх створило.

Подальша класифікація визначає вимоги щодо доступу. Застосуванню не потрібно виконувати аутентифікацію з токеном для перегляду «Відкритих об'єктів», але необхідно обов'язково виконати аутентифікацію з токеном для того, щоб переглянути «Приватні об'єкти», - користувач повинен пройти перевірку справжності/ аутентифікацію на токені шляхом введення PIN-коду або іншим методом (наприклад, використовуючи біометричні пристрої).

Токен може створювати і знищувати об'єкти, маніпулювати ними, і шукати їх. Він також може виконувати криптографічні функції з об'єктами.

«Атрибути» (Attributes) це характеристики, які відрізняють екземпляр об'єкта. У Cryptoki є загальні атрибути, наприклад, чи є об'єкт приватним/ особистим чи публічним/ відкритим. Є також атрибути, характерні для конкретного типу об'єктів, наприклад, довжина ключа для об'єкту «Ключ».

2.3. Користувачі

Ці Технічні специфікації визначають два типи користувачів (Users) токена:

«Офіцер безпеки» (Security Officer, SO);

«Звичайний користувач» (Normal User).

Тільки Звичайний користувач має право доступу до закритих об'єктів на токені, і доступ повинен надаватися лише після того, як Звичайний користувач був аутентифікований (пройшов перевірку справжності). Деякі токени можуть також вимагати, щоб користувач пройшов аутентифікацію перш, ніж можуть бути виконані на токені будь-які криптографічні функції з приватним об'єктами чи без них.

Роль SO полягає в ініціалізації токена та встановленні PIN-коду Звичайного користувача, і, можливо, маніпулюванні/ управлінні деякими відкритими об'єктами.

Звичайний користувач не може увійти в систему, поки SO не встановив PIN для цього Звичайного користувача.

Крім підтримки двох типів користувачів, Cryptoki не зачіпає відносини між SO і користувачами. Зокрема, SO і Звичайним користувачем може бути та сама особа, або можуть бути різні особи.

Cryptoki лише визначає, що PIN-код, які поставляються через застосування, є рядком символів змінної довжини із набору, наведеного у таблиці 2.1:

Таблиця 2.1. Набір символів

Категорія	Символи
Букви	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h i j k l m n o p q r s t u v w x y z
Числа	0 1 2 3 4 5 6 7 8 9
Графічні символи characters	! " # % & ' () * + , - . / : ; < = > ? [\] ^ _ { } ~
Пустий символ	' '

PIN-коди, які поставляються будь-яким іншим способом, окрім як через застосування (наприклад, PIN-коди, введені через PINpad на токени) не регламентуються у цих Технічних специфікаціях.

2.4. Символи та аббревіатури

У цих Технічних специфікаціях використовуються такі символи:

Таблиця 2.1. Символи

Символ	Опис
N/A	Не застосовується
R/O	Тільки читання
R/W	Читання/запис

У цих Технічних специфікаціях такі префікси:

Таблиця 2.3. Префікси

Префікс	Опис
C_	Функція
CK_	Тип даних або загальні константи
CKA_	Атрибут
CKF_	Біт прапора
CKO_	Клас об'єкту
CKR_	Повернене значення
CKU_	Тип користувача

Cryptoki базується на типах ANSI C і визначає такі дані:

```

/* без знакове 8-бітове значення */
typedef unsigned char CK_BYTE;

/* без знаковий 8-бітовий символ */
typedef CK_BYTE CK_CHAR;

/* 8-бітовий символ UTF-8 */
typedef CK_BYTE CK_UTF8CHAR;

/* Булевий/ логічний прапор */
typedef CK_BYTE CK_BBOOL;

/* без знакове значення, принаймні довжиною 32 біта*/
typedef unsigned long int CK_ULONG;

/* без знакове значення, того ж розміру, як CK_ULONG */
typedef long int CK_LONG;

/* знакові 32 біта; кожен біт – це Булевий/ Логічний прапор */
typedef CK_ULONG CK_FLAGS;

```

Cryptoki використовує покажчики (Pointer) на деякі типи даних, які залежать від реалізації. Це такі типи покажчиків:

```

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 */

```

Cryptoki також визначає покажчик на CK_VOID_PTR, який залежить від реалізації.

```

CK_VOID_PTR_PTR /* Pointer to a CK_VOID_PTR */

```

Cryptoki визначає покажчик C-style NULL, який відрізняється від будь-якого дійсного покажчика:

```

NULL_PTR      /* A NULL pointer */

```

Тип даних CK_UTF8CHAR містить символи Юнікоду в кодуванні UTF-8, визначені у RFC 2279 «UTF-8, a transformation format of ISO 10646». UTF-8 дозволяє інтернаціоналізувати при збереженні зворотної сумісності з визначенням PKCS#11.

У Cryptoki тип даних CK_BBOOL є Логічним типом Boolean, який може мати значення істина/true або хибний/false. Нульове значення означає значення false, а ненульове значення означає, що це - true. Аналогічно, одиничний біт прапора CKF_..., також може бути встановленим (true) або не встановленим

(false). Для зручності Cryptoki визначає такі макроси для використання із значеннями типу CK_BBOOL:

```
#define CK_FALSE 0
#define CK_TRUE 1
```

Для зворотної сумісності, файли заголовків для даної версії Cryptoki також визначає TRUE та FALSE як (CK_DISABLE_TRUE_FALSE може бути встановлено застосуванням постачальника):

```
#ifndef CK_DISABLE_TRUE_FALSE
#ifndef FALSE
#define FALSE CK_FALSE
#endif
```

```
#ifndef TRUE
#define TRUE CK_TRUE
#endif
#endif
```

2.5. Програми/Застосування та їх використання Cryptoki

Для Cryptoki програма (Application) складається з одного адресного простору і всі потоки (threads) управління працюють в ньому. Програма стає «Cryptoki застосуванням», викликавши функцію Cryptoki C_Initialize з одного із потоків програми, а після чого програма може викликати інші функції Cryptoki. Коли програма виконується за допомогою Cryptoki, вона викликає функцію Cryptoki C_Finalize і перестає бути застосуванням Cryptoki.

2.5.1. Програми та процеси

В цілому, на більшості платформ, попередній абзац означає, що програма складається з одного процесу.

Розглянемо UNIX процес P, який стає застосуванням Cryptoki шляхом виклику C_Initialize, а потім використовує системний виклик fork() для створення дочірнього процесу C. Так як P і C мають окремі адресні простори, вони не є частиною одного й того ж застосування. Тому, якщо процесу C необхідно використовувати Cryptoki, то він повинен виконати свій власний виклик C_Initialize. Крім того, якщо процесу C необхідно увійти до токена (ів), що буде доступний через Cryptoki, то він повинен увійти до цього, навіть якщо процес P вже увійшов до системи, так як P і C повністю окремі застосування.

В даному конкретному випадку (коли C є дочірнім процесом, який є застосуванням Cryptoki), поведінка Cryptoki не визначена, якщо C намагається використовувати його без власного виклику C_Initialize. В ідеалі, така спроба буде повертати значення CKR_CRYPTOKI_NOT_INITIALIZED, однак через спосіб fork() працює. Таким чином, поведінка Cryptoki у цій ситуації залишається невизначеною.

У сценарії, зазначеному вище, процес C повинен фактично викликати C_Initialize чи потрібно використання Cryptoki, чи ні, а потім повинен викликати

C_Finalize. Це (мати дочірній безпосередній виклик C_Initialize, а потім виклик C_Finalize, якщо батько використовує Cryptoki) вважається хорошою практикою програмування Cryptoki, так як може запобігти існуванню обірваних дубльованих ресурсів, які були створені під час виклику fork(), однак не вимагається Cryptoki.

2.5.2. Програми та потоки

Деякі програми будуть отримувати доступ до бібліотеки Cryptoki вбагато-поточних режимах. Cryptoki дозволяє застосуванням надавати інформацію в бібліотеки, так що вони можуть дати належну підтримку багато поточності. Зокрема, коли програма ініціалізує бібліотеку Cryptoki шляхом виклику C_Initialize, вона може вказати одну із чотирьох можливих поведінок багато-поточності для бібліотеки:

1. Програма може вказати, що вона не буде отримувати доступ до бібліотеки одночасно з декількох потоків, тому бібліотеці не потрібно турбуватися про виконання будь-якого типу блокування для безпеки потоків.

2. Програма може вказати, що вона буде отримувати доступ до бібліотеки одночасно з декількох потоків, і бібліотека повинна бути в змозі використати вбудовані (native) примітиви роботи системи синхронізації для забезпечення потоково-безпечної поведінки.

3. Програма може вказати, що вона буде отримувати доступ до бібліотеки одночасно з декількох потоків, і бібліотека повинна використовувати набір, що поставляється програмою, примітивів синхронізації для забезпечення потоково-безпечної поведінки.

4. Програма може вказати, що вона буде отримувати доступ до бібліотеки одночасно з декількох потоків, та бібліотека повинна використовувати або вбудовані у операційну систему примітиви синхронізації, або набір, що поставляється програмою, примітивів синхронізації для забезпечення потоково-безпечної поведінки

3 та 4 типи поведінки, зазначені вище підходять для багато-поточних застосувань, які не використовують вбудовані у операційну систему поточкові моделі. Примітиви синхронізації, що поставляються програмами, складаються з чотирьох функцій для управління mutex-об'єктами (mutex, м'ютекс, походить від *mutual exclusion* – взаємне виключення/ недопущення належності обом сторонам) в моделі потоків. М'ютекс об'єкти є простими об'єктами, які в будь-який момент часу можуть бути в двох станах: розблоковано або заблоковано. Якщо потоком здійснюється виклик на блокування м'ютекс-об'єкту, який вже заблокований, то потік блокується (чекає), поки м'ютекс-об'єкт не стане не заблокованим, тоді він блокує його і виклик повертається/ завершається. Якщо більш ніж один потік блокується на певному м'ютекс-об'єкті, і якщо м'ютекс буде розблоковано, то точно один з цих потоків отримає блокування м'ютекс-об'єкта і поверне управління до того, хто викликав (інші блокуючі потоки будуть продовжувати очікувати).

2.6. Сесії

Cryptoki вимагає, щоб застосування відкривало одну або кілька сесій (session) з токеном для отримання доступу до об'єктів і функцій токenu.

Сесія надає логічний зв'язок між застосуванням і токеном. Сесія може бути сесією «Читання / запису» (R/W session) або «Тільки для читання» (R/O session).

«Читання / запис» і «Тільки для читання» стосується доступу до об'єктів токenu, а не до об'єктів сесії. В обох типах сесії програма може створювати, читати, записувати і знищувати об'єкти сесії та читати об'єкти токenu. Однак, тільки в сесії «Читання / запису» застосування може створювати, змінювати і знищувати об'єкти токenu.

2.6.1. Сесія «Тільки для читання»

Сесія «Тільки для читання» може бути в одному з двох станів, як показано на рис.2.3. Коли сесія відкривається, то вона може знаходитися або в стані "R/O Public Session" (R/O відкрита сесія) стані (якщо програма не має раніше відкритих сесій, які увійшли до системи) або у стані "R/O User Functions" (R/O функцій користувача), якщо програма вже має відкриту сесію, яка увійшла до системи. Слід звернути увагу, що сесія «Тільки для читання» Офіцера безпеки SO не існує.

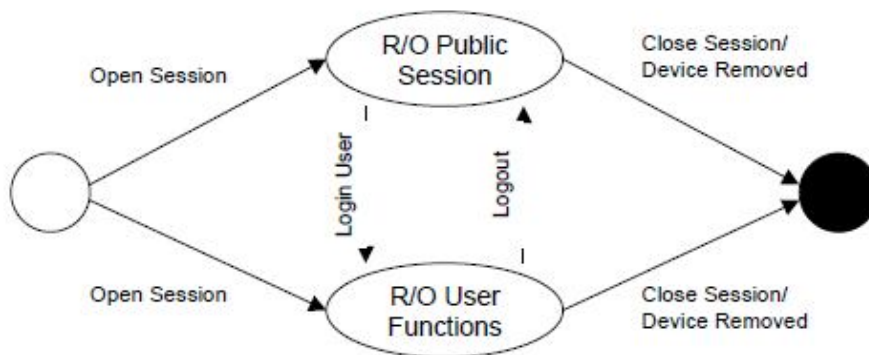


Рис. 2.3. Стан сесії «Тільки для читання»

Таблиця 2.4 ілюструє стани сесії «Тільки для читання»:

Таблиця 2.4. Стан сесії «Тільки для читання»

Стан	Опис
R/O Public Session	Застосування відкрило сесію «Тільки для читання». Програма має доступ тільки для читання до відкритих об'єктів токenu і доступ читання / запису до відкритих об'єктів сесії.
R/O User Functions	Звичайний користувач пройшов перевірку автентичності у токени. Програма має доступ тільки для читання до всіх об'єктів токenu (відкритих або закритих/ особистих) і доступ для читання / запису до всіх об'єктів сесії (відкритих або закритих/ особистих).

2.6.2. Сесія «Читання / запису»

Сесія «Читання / запису» може перебувати в одному з трьох станів, як показано на рис 2.4. При відкритті сесії вона знаходиться або у стані "R/W Public Session" стані (якщо програма не має раніше відкритих сесій, які увійшли в систему), у стані "R/W User Functions" (якщо програма вже має відкриту сесію, у якій зареєструвався звичайний користувач), або у стані "R/W SO Functions" стані (якщо програма вже має відкриту сесію, що відкрита SO).

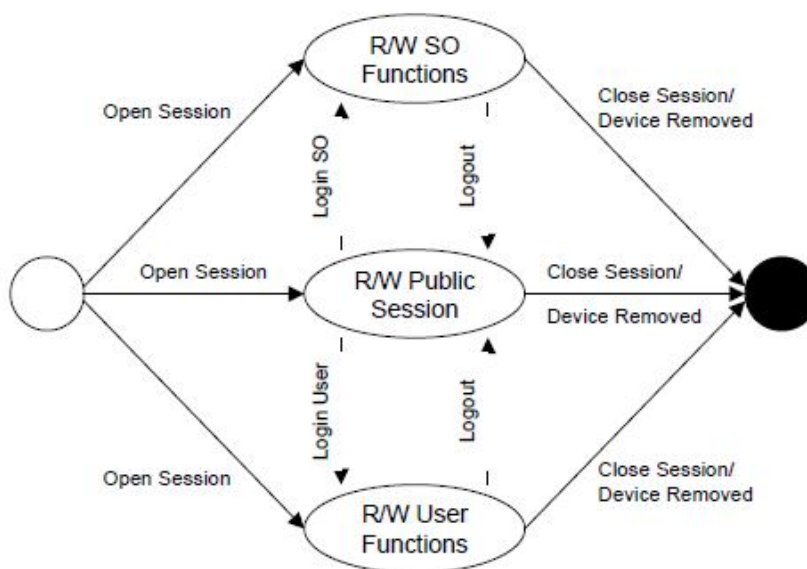


Рис. 2.4. Стан сесії «Читання / запису»

Таблиця 2.5 ілюструє стани сесії «Читання / запису»:

Таблиця 2.5. Стан сесії «Читання / запису»

Стан	Опис
R/W Public Session	Програма відкрила сесію «Читання / запису». Програма має доступ читання / запису до всіх відкритих об'єктів.
R/W SO Functions	Офіцер безпеки SO аутентифікувався у токени. Програма має доступ читання / запису тільки для відкритих об'єктів токена, але не для закритих. SO може встановити PIN звичайного користувача.
R/W User Functions	Звичайний користувач аутентифікувався у токени. Програма має доступ читання / запису до всіх об'єктів

2.6.3. Сесійні події

Сесійні події є причиною зміни стану сесії. Таблиця 2.5 описує сесійні події:

Таблиця 2.5. Сесійні події

Подія	Відбувається коли...
Аутентифікація Офіцера безпеки (Log In SO)	SO проходить аутентифікацію у токени
Аутентифікація Звичайного користувача (Log In User)	Звичайний користувач проходить аутентифікацію у токени
Вийти з (Log Out)	Застосування відключає поточного користувача (SO чи Звичайного користувача)
Закрити сесію (Close Session)	Застосування закриває сесію чи всі сесії
Пристрій вилучено (Device Removed)	Пристрій, що лежить в основі токена, був вилучений зі свого слоту

Коли пристрій видаляється, всі сеанси всіх застосувань повинні автоматично закриватися - Вийти з (Log Out) системи. Крім того, всі сесії будь-якої іншої програми, що працювала з пристроєм, закриваються – програма не може мати сесію з токеном, якого не існує. Реально, Cryptoki не може постійно стежити чи токен є присутнім, і тому відсутність токена, ймовірно, може бути не помічена, поки виконується функція Cryptoki.

У Cryptoki, всі сесії програми з токеном повинні мати однаковий login/logout статус (тобто для даних програми та токена повинна виконуватися одна з таких умов: всі сесії є відкритими сесіями (public sessions), всі сесії є SO сесіями; чи всі сесії є сесіями Користувача). Коли сесія програми входить до токена, всі сесії, які мала програма з токеном отримують стан входу до системи, а коли сесія програми виходить з токена, всі сесії, які має програма з токеном набувають стан Вийти з (Log Out) системи.

2.6.4. Дескриптор сесії та дескриптор об'єкту

Дескриптор сесії (Session handle) - це Cryptoki присвоєне значення, яке ідентифікує сесію; подібно до дескриптора файлу, і визначається для функції, щоб вказати, у якій сесії повинна діяти функція.

Всі потоки (threads) програми мають рівний доступ до всіх дескрипторів сесій. Тобто, все, що може бути досягнуто з даним дескриптором файлу одним потоком програми, може також бути виконано з дескриптором файлу іншим потоком тієї ж програми.

Cryptoki також має дескриптор об'єкту (Object handle), який є ідентифікатором, що використовується для маніпулювання Cryptoki об'єктами. Дескриптори об'єкту подібні дескрипторам сесії в тому, що видимість даного об'єкта через дескриптор об'єкта є однаковою серед усіх потоків цього застосування.

Валідні дескриптор сесії та об'єкта в Cryptoki завжди повинні мати ненульові значення.

Для зручності розробників, Cryptoki визначає наступне символічне значення невалідності дескриптора:

CK_INVALID_HANDLE

2.6.5. Можливості сесій

Є три основних види операцій по відкриттю сесії, що може бути використаний для виконання:

- адміністративні операції (наприклад, вхід до системи);
- операції управління об'єктами (наприклад, створення або знищення об'єкта на токени), а також
- криптографічні операції (такі, як обчислення хеш-значення повідомлення).

Криптографічній операції іноді потрібно більше одного виклику функції Cryptoki API, щоб завершити виконання. Загалом, одна сесія може виконувати тільки одну операцію одночасно. З цієї причини, може бути бажано для однієї програми, щоб відкрити кілька сесій з одним токеном. Заради ефективності, однак, одна сесія на токени може виконувати такі пари типів операцій одночасно: обчислення хеш значення повідомлення та його зашифрування (цифровий підпис), розшифрування (перевіряння цифрового підпису) і обчислення хеш значення повідомлення; підписування або обчислення MAC і шифрування, а також розшифрування та перевіряння підписів або MAC. Подробиці про проведення одночасних криптографічних операцій в одній сесії, наводяться далі в цих Технічних специфікаціях.

Наслідком того, що в одній сесії можна, загалом, виконувати тільки одну операцію одночасно є те, що програма не може зробити кілька одночасних викликів функцій Cryptoki, які використовують спільну сесію. Якщо кілька потоків програми намагаються використати спільну сесію одночасно, то Cryptoki не може визначити, що відбувається. Отже, якщо кільком потокам програми потрібно використовувати Cryptoki, щоб отримати доступ до певного дескриптора, то доцільно кожному потоку мати свою власну сесію з дескриптором. Це справедливо незалежно від того, чи ні Cryptoki бібліотека ініціалізується в режимі, який дозволяє безпечний багато-потоківий доступ до неї. Навіть якщо це безпечно щодо доступу до бібліотеки з декількох потоків одночасно, все ще не завжди безпечно щодо використання конкретної сесії з декількох потоків одночасно.

2.7. Загальні типи даних

2.7.1. Загальна інформація

Cryptoki повинен надавати загальну інформацію такими типами:

- ◆ CK_VERSION; CK_VERSION_PTR

CK_VERSION є структурою, яка описує версію інтерфейсу Cryptoki, бібліотеки Cryptoki чи SSL реалізації, або версію апаратного чи програмного забезпечення слота або токена. Визначається таким чином:

```
typedef struct CK_VERSION {  
    CK_BYTE major;  
    CK_BYTE minor;  
} CK_VERSION;
```

Поля структури мають таке значення:

major – старший/ основний номер версії (ціла частина версії)

minor – молодший/ другорядний/ допоміжний номер версії (соті частини версії).

Приклад: Для версії 1.0, основний =1 і другорядний = 0. Для версії 2.10, основний = 2 і другорядний = 10.

У таблиці нижче зазначено основні і допоміжні значення версії для офіційно опублікованих специфікації Cryptoki PKCS#11.

Таблиця 2.6. Офіційні версії Cryptoki

Версія	Номер основний	Номер другорядний
1.0	0x01	0x00
2.01	0x02	0x01
2.10	0x02	0x0a
2,11	0x02	0x0B
2.20	0x02	0x14

Зміни в другорядних Cryptoki завжди сумісні зверху, в межах одного основного номеру версії.

CK_VERSION_PTR є покажчиком на CK_VERSION.

Cryptoki надає загальну інформацію такими типами:

♦ CK_INFO; CK_INFO_PTR

CK_INFO надає загальну інформацію про Cryptoki. Вона визначається таким чином:

```
typedef struct CK_INFO {  
    CK_VERSION cryptokiVersion;  
    CK_UTF8CHAR manufacturerID[32];  
    CK_FLAGS flags;  
    CK_UTF8CHAR libraryDescription[32];  
    CK_VERSION libraryVersion;  
} CK_INFO;
```

Поля структури мають такі значення:

cryptokiVersion - номер версії інтерфейсу Cryptoki для сумісності з наступними виправленнями цього інтерфейсу;
manufacturerID - ID виробника бібліотеки Cryptoki. Має бути доповненим порожнім символом (' '). *Не повинно* бути нульовим;
flags – біт прапорів, які відведені для майбутніх версій. Повинні бути нульовим для цієї версії;
libraryDescription - текстовий опис бібліотеки. Має бути доповнений порожнім символом (' '). *Не повинно* бути нульовим;
libraryVersion - номер версії бібліотеки Cryptoki.

Для бібліотек, написаних відповідно до цих Технічних специфікацій, значення *cryptokiVersion* повинно відповідати версії «2.20»; значення *libraryVersion* є номером версії самої бібліотеки програмного забезпечення.

CK_INFO_PTR є покажчик (pointer) для CK_INFO.

Cryptoki надає загальну інформацію такими типами:

◆ CK_NOTIFICATION

CK_NOTIFICATION утримує типи повідомлень/ сповіщень (notification), які Cryptoki надає програмам. Визначається таким чином:

```
typedef CK_ULONG CK_NOTIFICATION;
```

За цією версією Cryptoki, визначаються такі типи повідомлень:

CKN_SURRENDER

Повідомлення мають таке значення:

CKN_SURRENDER – Cryptoki передає виконання функції в сесії, з тим, що програма може виконувати інші операції. Після виконання будь-якої бажаної операції, програма повинна вказати Cryptoki чи продовжити, чи скасувати функцію.

2.7.2. Типи слоту та токenu

Cryptoki повинен надавати інформацію про слот, використовуючи такі типи:

◆ CK_SLOT_ID; CK_SLOT_ID_PTR

CK_SLOT_ID є присвоєним значенням Cryptoki, яке визначає ідентифікатор/ номер слоту. Він визначається таким чином:

```
typedef CK_ULONG CK_SLOT_ID;
```

Список ідентифікаторів CK_SLOT_IDs повертається функцією C_GetSlotList.

Апріорі, будь-яке значення CK_SLOT_ID може бути дійсним ідентифікатором слоту – зокрема, система може мати слот, що визначений значенням 0. Не обов'язково, що буде такий слот.

CK_SLOT_ID_PTR – це покажчик на CK_SLOT_ID.

Безпосередньо інформація про слот надається, використовуючи такі типи:

◆ CK_SLOT_INFO; CK_SLOT_INFO_PTR

CK_SLOT_INFO надає інформацію про слот. Визначається таким чином:

```
typedef struct CK_SLOT_INFO {
    CK_UTF8CHAR slotDescription[64];
    CK_UTF8CHAR manufacturerID[32];
    CK_FLAGS flags;
    CK_VERSION hardwareVersion;
    CK_VERSION firmwareVersion;
} CK_SLOT_INFO;
```

Поля структури мають такі значення:

slotDescription - текстовий опис слоту. Повинен бути доповнений порожнім символом (' '). Значення *не повинно* бути нульовим;

manufacturerID- ID виробника слоту. Повинен бути доповнений порожнім символом (' '). Значення *не повинно* бути нульовим;

flags – біти прапорів, які визначають характеристики слоту. Прапори, описані нижче (див. Таблиця 2.6).

hardwareVersion- номер версії апаратного забезпечення слоту;

firmwareVersion- номер версії вбудованого ПЗ слоту.

Таблиця 2.7 описує поля прапорів:

Таблиця 2.7. Відомості про прапори слоту

Біт прапора	Маска	Опис
CKF_TOKEN_PRESENT	0x00000001	True, якщо токен присутній в слоті (наприклад, пристрій знаходиться в зчитувачі)
CKF_REMOVABLE_DEVICE	0x00000002	True, якщо зчитувач підтримує змінні пристрої
CKF_HW_SLOT	0x00000004	True, якщо слот є слотом апаратного забезпечення, а не слотом програмного забезпечення реалізованого в "м'якому токені"/ програмному токені

Для заданого слоту значення прапора CKF_REMOVABLE_DEVICE ніколи не змінюється. Крім того, якщо цей прапор встановлено для даного слоту, то

прапор CKF_TOKEN_PRESENT цього слоту *завжди* буде встановлено. Тобто, якщо слот не підтримує змінний пристрій, то цей слот завжди містить токен.

CK_SLOT_INFO_PTR — це вказівник на CK_SLOT_INFO.

Cryptoki повинен надавати інформацію про токен, використовуючи такі типи:

◆ CK_TOKEN_INFO; CK_TOKEN_INFO_PTR

CK_TOKEN_INFO надає інформацію про токен. Вона визначається таким чином:

```
typedef struct CK_TOKEN_INFO {  
    CK_UTF8CHAR label[32];  
    CK_UTF8CHAR manufacturerID[32];  
    CK_UTF8CHAR model[16];  
    CK_CHAR serialNumber[16];  
    CK_FLAGS flags;  
    CK_ULONG ulMaxSessionCount;  
    CK_ULONG ulSessionCount;  
    CK_ULONG ulMaxRwSessionCount;  
    CK_ULONG ulRwSessionCount;  
    CK_ULONG ulMaxPinLen;  
    CK_ULONG ulMinPinLen;  
    CK_ULONG ulTotalPublicMemory;  
    CK_ULONG ulFreePublicMemory;  
    CK_ULONG ulTotalPrivateMemory;  
    CK_ULONG ulFreePrivateMemory;  
    CK_VERSION hardwareVersion;  
    CK_VERSION firmwareVersion;  
    CK_CHAR utcTime[16];  
} CK_TOKEN_INFO;
```

Поля структури мають такі значення:

Label – визначена програмою мітка, призначається під час ініціалізації токена. Повинна бути доповнена порожнім символом (‘ ‘). Значення *не повинно* бути нульовим.

manufacturerID - код виробника пристрою. Повинен бути доповнений порожнім символом (‘ ‘). Значення *не повинно* бути нульовим.

Model – модель пристрою. Повинна бути доповнена порожнім символом (‘ ‘). Значення *не повинно* бути нульовим.

serialNumber – текстовий серійний номер пристрою. Повинен бути доповнений порожнім символом (‘ ‘). Значення *не повинно* бути нульовим.

Flags – біт прапорів із зазначенням характеристик і стану пристрою як зазначено нижче (див Таблиця 2.7).

ulMaxSessionCount – максимальна кількість сесій, які можуть бути відкриті з токеном одночасно однією програмою (див. Примітку нижче).

ulSessionCount – кількість сесій, які ця програма в даний час відкрила з токеном (див. примітку нижче).

ulMaxRwSessionCount – максимальна кількість сесій читання/запису, які можуть бути відкриті з токеном одночасно однією програмою (див. примітку нижче).

ulRwSessionCount – число сесій читання/запису, які в даний час відкриті з токеном одночасно (див. примітку нижче).

ulMaxPinLen – максимальна довжина в байтах PIN-коду.

ulMinPinLen – мінімальна довжина в байтах PIN-коду.

ulTotalPublicMemory – загальний обсяг пам'яті токена у байтах, у якій можуть зберігатися відкриті об'єкти (див. примітку нижче).

ulFreePublicMemory – обсяг вільної пам'яті (не використаної) токена у байт для відкритих об'єктів (див. примітку нижче).

ulTotalPrivateMemory – загальний обсяг пам'яті токена у байтах, у якій можуть зберігатися закриті/ приватні об'єкти (див. примітку нижче).

ulFreePrivateMemory – обсяг вільної пам'яті (не використаної) токена у байтах для закритих/ приватних об'єктів (див. примітку нижче).

hardwareVersion – номер версії апаратного забезпечення.

firmwareVersion – номер версії вбудованого ПЗ.

utcTime – поточний час як рядок символів довжиною 16, представлений у форматі YYYYMMDDhhmmssxx (4 символи року; кожні 2 символи для місяця, дня, години, хвилини та секунди; і 2 додаткові зарезервовані символи '0'). Значення цього поля має сенс тільки для токенів, що оснащені годинником, як зазначено в інформації прапора у токені (див. Таблиця 2.7).

Таблиця 2.8 описує поля прапорів:

Таблиця 2.8. Відомості про прапори токенау

Біт прапора	Маска	Опис
CKF_RNG	0x00000001	True, якщо токен має власний генератор випадкових чисел
CKF_WRITE_PROTECTED	0x00000002	True, якщо токен захищено від запису (див. нижче)
CKF_LOGIN_REQUIRED	0x00000004	True, якщо є деякі криптографічні

Біт прапора	Маска	Опис
		функції, які користувач повинен виконувати.
CKF_USER_PIN_INITIALIZED	0x00000008	True, якщо ініціалізовано PIN звичайного користувача
CKF_RESTORE_KEY_NOT_NEEDED	0x00000020	True, якщо успішно збережено стан криптографічних операцій сесії, <i>завжди</i> містить всі ключі, які необхідно відновити у стані сесії
CKF_CLOCK_ON_TOKEN	0x00000040	True, якщо токен має свій власний годинник
CKF_PROTECTED_AUTHENTICATION_PATH	0x00000100	True, якщо токен має «захищений шлях аутентифікації», за допомогою якого користувач може увійти в токен без застосування PIN через бібліотеку Cryptoki
CKF_DUAL_CRYPTO_OPERATIONS	0x00000200	True, якщо одна сесія з токеном може виконувати подвійні криптографічні операції
CKF_TOKEN_INITIALIZED	0x00000400	True, якщо токен ініціалізовано за допомогою C_InitializeToken, або еквівалентного

Біт прапора	Маска	Опис
		механізму, що виходить за рамки цієї Специфікації. Виклик C_InitializeToken, коли встановлено цей прапор, призведе до повторної ініціалізації токена
CKF_SECONDARY_AUTHENTICATION	0x00000800	True, якщо токен підтримує вторинну автентифікацію об'єкту закритого ключа. (Відмінено; нові реалізації не повинні встановлювати цей прапор)
CKF_USER_PIN_COUNT_LOW	0x00010000	True, якщо неправильний PIN користувача введено принаймні один раз, починаючи з останньої успішної автентифікації.
CKF_USER_PIN_FINAL_TRY	0x00020000	True, якщо наданий неправильний PIN користувача буде заблоковано.
CKF_USER_PIN_LOCKED	0x00040000	True, якщо заблоковано PIN користувача. Вхід користувача у токен буде неможливим.
CKF_USER_PIN_TO_BE_CHANGED	0x00080000	True, якщо

Біт прапора	Маска	Опис
		значення PIN користувача є значенням за замовчанням, встановленим при ініціалізації або виробництві, або PIN вже не дійсний.
CKF_SO_PIN_COUNT_LOW	0x00100000	True, якщо неправильно введено SO PIN принаймні один раз, починаючи з останньої успішної автентифікації.
CKF_SO_PIN_FINAL_TRY	0x00200000	True, якщо надано неправильний SO PIN буде заблоковано.
CKF_SO_PIN_LOCKED	0x00400000	True, якщо SO PIN заблоковано. Вхід користувача в токен буде неможливим.
CKF_SO_PIN_TO_BE_CHANGED	0x00800000	True, якщо значення SO PIN є значенням за замовчанням, встановленим при ініціалізації або виробництві, або PIN є вже не дійсний.

CK_TOKEN_INFO_PTR є покажчиком на CK_TOKEN_INFO.

2.7.3. Типи сесії

Cryptoki надає інформацію сесії, використовуючи такі типи:

◆ CK_SESSION_HANDLE; CK_SESSION_HANDLE_PTR

CK_SESSION_HANDLE є присвоєним значення Cryptoki, яке визначає сесію. Визначається таким чином:

```
typedef CK_ULONG CK_SESSION_HANDLE;
```

Дійсні дескриптори сесії у Cryptoki завжди мають ненульові значення. Для зручності розробників, Cryptoki визначає таке символічне значення недійсного дескриптора:

```
CK_INVALID_HANDLE
```

CK_SESSION_HANDLE_PTR є покажчиком на CK_SESSION_HANDLE.

Cryptoki надає інформацію про тип користувачів, використовуючи такі типи:

◆ CK_USER_TYPE

CK_USER_TYPE містить типи користувачів Cryptoki, що описані далі. Визначається таким чином:

```
typedef CK_ULONG CK_USER_TYPE.
```

Для цієї версії Cryptoki визначаються такі типи користувачів:

```
CKU_SO
```

```
CKU_USER
```

```
CKU_CONTEXT_SPECIFIC.
```

Cryptoki надає інформацію про стан/ статус сесії, використовуючи такі типи:

◆ CK_STATE

CK_STATE утримує стан сесії. Це визначається таким чином:

```
typedef CK_ULONG CK_STATE;
```

Для цієї версії Cryptoki, визначені такі стани сесії:

```
CKS_RO_PUBLIC_SESSION
```

```
CKS_RO_USER_FUNCTIONS
```

```
CKS_RW_PUBLIC_SESSION
```

```
CKS_RW_USER_FUNCTIONS
```

```
CKS_RW_SO_FUNCTIONS
```

Cryptoki надає інформацію про сесію, використовуючи такі типи:

◆ CK_SESSION_INFO; CK_SESSION_INFO_PTR

CK_SESSION_INFO надає інформацію про сесію. Визначається таким чином:

```
typedef struct CK_SESSION_INFO {
```

```
    CK_SLOT_ID slotID;
```

```
    CK_STATE state;
```

```

CK_FLAGS flags;
CK_ULONG ulDeviceError;
} CK_SESSION_INFO;

```

Поля структури мають такі значення:

slotID – ідентифікатор (ID) слоту, який взаємодіє з токеном.

State – стан сесії.

Flags- біт прапорів, який визначає тип сесії; прапори визначено нижче.

ulDeviceError - код помилки, визначається криптографічним пристроєм.

Використовується для помилок, які не охоплені Cryptoki.

Таблиця 2.9 описує поля прапорів:

Таблиця 2.9. Відомості про прапори сесії

Біт прапора	Маска	Опис
CKF_RW_SESSION	0x00000002	True, якщо сесія для читання/запису; false, якщо сесія тільки для читання
CKF_SERIAL_SESSION	0x00000004	Цей прапор надається для зворотної сумісності і завжди повинен бути встановлений у True

CK_SESSION_INFO_PTR є покажчиком на CK_SESSION_INFO.

2.7.4. Типи об'єктів

Cryptoki надає інформацію про дескриптор об'єкту:

◆ CK_OBJECT_HANDLE

Визначається таким чином:

```
typedef CK_ULONG CK_OBJECT_HANDLE;
```

Коли об'єкт створюється або знаходиться програмою на токени, Cryptoki призначає дескриптор об'єкту для цієї сесії програми, з метою використання під час доступу до нього. Певному об'єкту або токени не обов'язково мати покажчик, який встановлюється на протязі існування цього об'єкта; однак, якщо дана сесія може використовувати певний дескриптор для доступу до конкретного об'єкта, то ця сесія буде продовжувати використовувати цей дескриптор для доступу до цього об'єкта, оскільки сесія продовжує існувати, об'єкт продовжує існувати і об'єкт продовжує бути доступним для сесії.

Допустимий дескриптор об'єкту у Cryptoki завжди має ненульове значення. Для зручності розробників, Cryptoki визначає такі недопустимі значення дескриптору:

CK_INVALID_HANDLE

CK_OBJECT_HANDLE_PTR є покажчиком на CK_OBJECT_HANDLE.

Cryptoki надає інформацію про тип апаратних особливостей:

◆ CK_HW_FEATURE_TYPE

CK_HW_FEATURE_TYPE – це значення, яке визначає тип паратних особливостей пристрою. Воно визначається таким чином:

```
typedef CK_ULONG CK_HW_FEATURE_TYPE;
```

Типи особливостей обладнання визначаються об'єктами, які їх використовують. Тип специфіковано по об'єктах через SKA_HW_FEATURE_TYPE атрибут об'єкта.

Значення, визначені постачальником, також можуть бути вказані для цього типу:

CKH_VENDOR_DEFINED

Тип CKH_VENDOR_DEFINED зарезервовано для постачальників токена. Для сумісності, виробники повинні реєструвати свої типи об'єктів через процес PKCS.

Cryptoki надає інформацію про тип ключа:

◆ CK_KEY_TYPE

CK_KEY_TYPE – цей значення, яке визначає типа ключа. Визначається таким чином:

```
typedef CK_ULONG CK_KEY_TYPE;
```

Основні типи визначаються з об'єктами та механізмами, які їх використовують. Тип ключа специфіковано у об'єктах через SKA_KEY_TYPE атрибут об'єкта.

Певні значення постачальника для цього типу також можуть бути вказані.

CKK_VENDOR_DEFINED

Тип CKK_VENDOR_DEFINED зарезервовано для постачальників токена. Для сумісності, виробники повинні реєструвати свої основні типи через процес PKCS.

Cryptoki надає інформацію про тип сертифікату:

◆ CK_CERTIFICATE_TYPE

CK_CERTIFICATE_TYPE – це значення, яке ідентифікує тип сертифікату, та визначається таким чином:

```
typedef CK_ULONG CK_CERTIFICATE_TYPE;
```

Тип Сертифікат визначається з об'єктами і механізмами, які його використовують. Тип Сертифікат специфікується через `CKA_CERTIFICATE_TYPE` атрибут об'єкта.

Певні значення для цього типу також можуть бути вказані Постачальником.
`CKC_VENDOR_DEFINED`

Тип Сертифікату `CKC_VENDOR_DEFINED` зарезервовано постачальників токена. Для сумісності, виробники повинні реєструвати свої типи сертифікатів через процес PKCS.

Cryptoki надає інформацію про атрибути об'єкту, використовуючи типи:

◆ `CK_ATTRIBUTE`; `CK_ATTRIBUTE_PTR`

`CK_ATTRIBUTE` це структура, що містить тип, значення та довжину атрибута. Вона визначається таким чином:

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

Поля структури мають такі значення:

Type – тип атрибута;

pValue – покажчик на значення атрибута;

ulValueLen – довжина значення у байтах.

Якщо атрибут не має ніякого значення, то *ulValueLen* = 0 і значення *pValue* не має значення.

Масив атрибутів `CK_ATTRIBUTES` називається «шаблоном» і використовується для створення, роботи та пошуку об'єктів. Порядок атрибутів у шаблоні, не має значення, навіть якщо шаблон містить атрибути виробника. Зауважте, що *pValue* є пустим/порожнім покажчиком, який полегшує передачу довільних значень. Програма та Cryptoki бібліотека повинні гарантувати, що покажчик можна безпечно присвоювати до очікуваного типу, тобто без помилок щодо вирівнювання слів (*word-alignment*).

`CK_ATTRIBUTE_PTR` це вказівник на `CK_ATTRIBUTE`.

Cryptoki надає інформацію про дату, використовуючи типи:

`CK_DATE`

`CK_DATE` is a structure that defines a date. It is defined as follows:

```
typedef struct CK_DATE {  
    CK_CHAR year[4];
```

```
CK_CHAR month[2];
CK_CHAR day[2];
} CK_DATE;
```

Cryptoki надає інформацію про Логічні об'єкти, використовуючи типи:

- ◆ CK_TRUE – значення, яке може приймати атрибут, що відповідає істині.
- ◆ CK_FALSE – значення, яке може приймати атрибут, що відповідає брехні.

2.7.5. Типи даних для механізму

Cryptoki повинен підтримувати такі типи для опису типу механізмів:

- ◆ CK_MECHANISM_TYPE; CK_MECHANISM_TYPE_PTR

CK_MECHANISM_TYPE це значення, яке визначає тип механізму. Воно визначається таким чином:

```
typedef CK_ULONG CK_MECHANISM_TYPE
```

Типи механізмів визначаються описом об'єктів та механізмів, які їх використовують. Для цього типу також можуть бути специфіковані значення, визначені постачальником:

```
CKM_VENDOR_DEFINED
```

Типи механізмів CKM_VENDOR_DEFINED остаточно зарезервовані для постачальника токена. Для забезпечення оперативної сумісності, постачальники повинні зареєструвати їх типи механізмів через PKCS процес.

CK_MECHANISM_TYPE_PTR це покажчик на CK_MECHANISM_TYPE.

Cryptoki підтримує такі типи для опису механізмів:

- ◆ CK_MECHANISM; CK_MECHANISM_PTR

CK_MECHANISM є структурою, яка специфікує конкретний механізм і будь-які параметри, що є необхідними. Визначається таким чином:

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

Поля структури мають такі значення:

Mechanism - типу механізму;

pParameter – покажчик на параметр, який потрібен механізму;

ulParameterLen - довжина параметра в байтах.

Зауважте, що *pParameter* є пустим/порожнім покажчиком, який полегшує передачу довільних значень. Програма та Cryptoki бібліотека повинні гарантувати, що покажчик можна безпечно присвоювати до очікуваного типу, тобто без помилок щодо вирівнювання слів (*word-alignment*).

CK_MECHANISM_PTR це вказівник на CK_MECHANISM.

Cryptoki підтримує такі типи для опису інформації про механізм:

♦ CK_MECHANISM_INFO; CK_MECHANISM_INFO_PTR

CK_MECHANISM_INFO це структура, що надає інформацію про конкретний механізм. Визначається таким чином:

```
typedef struct CK_MECHANISM_INFO {
    CK_ULONG ulMinKeySize;
    CK_ULONG ulMaxKeySize;
    CK_FLAGS flags;
} CK_MECHANISM_INFO;
```

Поля структури мають такі значення:

ulMinKeySize – мінімальний розмір ключа для механізму (від механізму залежить, чи це вимірюється у бітах, чи у байтах);

ulMaxKeySize – максимальний розмір ключа для механізму (від механізму залежить, чи це вимірюється у бітах, чи у байтах);

flags – біт прапорів визначає можливості механізму.

Для деяких механізмів значення полів *ulMinKeySize* і *ulMaxKeySize* не мають сенсу.

Таблиця 2.10 описує поля прапорів:

Таблиця 2.10. Прапори інформації про механізм

Біт прапора	Маска	Опис
CKF_HW	00x00000001	True, якщо механізм виконується пристроєм; false, якщо механізм виконується програмно (поза межами пристрою)
CKF_ENCRYPT	00x00000100	True, якщо механізм може бути використаний з C_EncryptInit
CKF_DECRYPT	00x00000200	True, якщо механізм може бути використаний з C_DecryptInit
CKF_DIGEST	00x00000400	True, якщо механізм може бути використаний з C_DigestInit
CKF_SIGN	00x00000800	True, якщо механізм може бути використаний з C_SignInit
CKF_SIGN_RECOVER	00x00001000	True, якщо механізм може бути

Біт прапора	Маска	Опис
		використаний з C_SignRecoverInit
CKF_VERIFY	00x00002000	True, якщо механізм може бути використаний з C_VerifyInit
CKF_VERIFY_RECOVER	00x00004000	True, якщо механізм може бути використаний з C_VerifyRecoverInit 3
CKF_GENERATE	00x00008000	True, якщо механізм може бути використаний з C_GenerateKey
CKF_GENERATE_KEY_PAIR	00x00010000	True, якщо механізм може бути використаний з C_GenerateKeyPair 3
CKF_WRAP	00x00020000	True, якщо механізм може бути використаний з C_WrapKey
CKF_UNWRAP	00x00040000	True, якщо механізм може бути використаний з C_UnwrapKey
CKF_DERIVE	00x00080000	True, якщо механізм може бути використаний з C_DeriveKey
CKF_EXTENSION	00x80000000	True, якщо є розширення прапорів; false, якщо немає розширень. Має бути false для цієї версії.

CK_MECHANISM_INFO_PTR є покажчиком на CK_MECHANISM_INFO.

У випадку апаратної підтримки механізму, у описі механізму (структура «CK_MECHANISM_INFO») в полі «Flags» повинен бути встановлений прапорець «CKF_HW».

2.7.6. Типи для функцій

Cryptoki надає інформацію про функції з такими типами даних:

◆ CK_RV

CK_RV визначає значення, що повертається функцією Cryptoki. Визначається таким чином:

```
typedef CK_ULONG CK_RV
```

Значення, визначені постачальником для цього типу, також можуть бути специфіковані:

```
CKR_VENDOR_DEFINED
```

Повернені значення CKR_VENDOR_DEFINED зарезервовані для постачальників токени. Для забезпечення оперативної сумісності, постачальники повинні зареєструвати їх повернені значення через PKCS процес.

Cryptoki підтримує такі типи:

◆ CK_NOTIFY

CK_NOTIFY це тип покажчика на функцію, що використовується Cryptoki для виконання сповіщення про зворотній виклик. Визначається таким чином:

```
typedef CK_CALLBACK_FUNCTION(CK_RV, CK_NOTIFY)(  
    CK_SESSION_HANDLE hSession,  
    CK_NOTIFICATION event,  
    CK_VOID_PTR pApplication  
);
```

Аргументи функції сповіщення зворотного виклику, мають такий зміст:

hSession – дескриптор сесії, що виконує зворотній виклик;

Event – тип сповіщення зворотного виклику;

pApplication – значення, що залежить від програми. Це те ж саме значення, яке було прийнято C_OpenSession для відкриття сесії виконання зворотного виклику.

Cryptoki підтримує список функцій, використовуючи такі типи:

◆ CK_FUNCTION_LIST; CK_FUNCTION_LIST_PTR;
CK_FUNCTION_LIST_PTR_PTR

CK_FUNCTION_LIST є структурою, що містить версію Cryptoki та покажчик функції для кожної функції у Cryptoki API. Вона визначається наступним чином:

```
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;

```

Кожна бібліотека Cryptoki має статичну структуру CK_FUNCTION_LIST, і покажчик на неї (або його копію, яка належить також бібліотеці) може бути отримано функцією C_GetFunctionList. Значення, на яке цей покажчик вказує, використовується програмою для швидкого пошуку місця розташування коду виконання для кожної функції у Cryptoki API.

Кожна функція в Cryptoki API повинна мати точку входу, визначену в структурі бібліотеки Cryptoki CK_FUNCTION_LIST.

Якщо певна функція у Cryptoki API не підтримується бібліотекою, то покажчик на цю функцію в структурі бібліотеки CK_FUNCTION_LIST повинен вказувати на заглушку (stub) функції, що повертає CKR_FUNCTION_NOT_SUPPORTED.

Програмою може або не може бути змінена статична структура CK_FUNCTION_LIST бібліотеки Cryptoki. Незалежно від цього, програма ніколи не повинна намагатися зробити це.

CK_FUNCTION_LIST_PTR це покажчик CK_FUNCTION_LIST.

CK_FUNCTION_LIST_PTR_PTR це покажчик на CK_FUNCTION_LIST_PTR.

2.7.7. Типи, пов'язані з блокуванням

Типи, пов'язані з блокуванням (Locking-related), у цьому розділі надаються виключно для програм, яким необхідно отримати доступ до Cryptoki з декількох потоків одночасно. Програмам, які не будуть цього робити, не потрібно використовувати будь-який з цих типів.

Див. також розділ 2.5.2 про м'ютекс-об'єкти.

◆ CK_CREATEMUTEX

CK_CREATEMUTEX це тип покажчика на функцію, що поставляються програмою, яка створює новий м'ютекс-об'єкт і повертає покажчик на нього. Визначається таким чином:

```
typedef CK_CALLBACK_FUNCTION(CK_RV, CK_CREATEMUTEX) (
    CK_VOID_PTR_PTR ppMutex
);
```

Виклик функції CK_CREATEMUTEX повертає покажчик на новий об'єкт у місці, на яке вказує ppMutex.

Така функція повинна повертати одне з наступних значень: CKR_OK, CKR_GENERAL_ERROR, CKR_HOST_MEMORY.

◆ CK_DESTROYMUTEX

CK_DESTROYMUTEX це тип покажчика на функцію, що поставляється програмою, яка руйнує існуючий м'ютекс-об'єкт. Визначається таким чином:

```
typedef CK_CALLBACK_FUNCTION(CK_RV, CK_DESTROYMUTEX) (
    CK_VOID_PTR pMutex
);
```

Аргумент функції CK_DESTROYMUTEX є покажчиком на об'єкт, який повинен бути знищений. Така функція повинна повертати одне з наступних значень: CKR_OK, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MUTEX_BAD.

◆ CK_LOCKMUTEX і CK_UNLOCKMUTEX

CK_LOCKMUTEX - це тип покажчика на функцію, що поставляється програмою, яка блокує існуючий м'ютекс-об'єкт. CK_UNLOCKMUTEX це тип покажчика на функцію, що поставляється програмою, яка розблоковує існуючий м'ютекс-об'єкт. Належна поведінка для цих типів функцій виглядає наступним чином:

- Якщо функція CK_LOCKMUTEX викликається для об'єкта, який не заблоковано, то той, що викликає потік, отримує блокування для цього м'ютекса і функція повертає управління.
- Якщо функція CK_LOCKMUTEX викликається для м'ютекс-об'єкта іншим потоком, ніж той, що його заблокував, то цей потік чекає поки м'ютекс-об'єкт буде розблоковано.
- Якщо функція CK_LOCKMUTEX викликається для м'ютекс-об'єкта, заблокованого цим же потоком (який заблокував), поведінка функції не визначена.
- Якщо функція CK_UNLOCKMUTEX викликається для м'ютекс-об'єкта, заблокованого цим же потоком (який заблокував), то м'ютекс-об'єкт розблокується і повертається до функції виклику. Крім того:
 - Якщо рівно один потік був заблокований на цьому конкретному м'ютекс-об'єкті, то цей потік зупиняється, отримує (нове) блокування для цього м'ютекса, і йому повертається CK_LOCKMUTEX виклик.

- Якщо більш ніж один потік був заблокований на цьому конкретному м'ютекс-об'єкті, то точно один з заблокованих потоків обирається якимось чином. Потік, який обрано, зупиняється, отримує (нове) блокування м'ютекса, і йому повертається CK_LOCKMUTEX виклик. Всі інші потоки блокування для цього конкретного м'ютекса продовжують блокувати.
- Якщо функція CK_UNLOCKMUTEX викликається для м'ютекс-об'єкта, який не заблоковано, то виклик функції повертає код помилки CKR_MUTEX_NOT_LOCKED.
- Якщо функція CK_UNLOCKMUTEX викликається для м'ютекс-об'єкта, заблокованого якимось потоком, іншим, ніж потік, що викликає, то поведінка виклику функції не визначена.

CK_LOCKMUTEX визначається таким чином:

```
typedef CK_CALLBACK_FUNCTION (CK_RV, CK_LOCKMUTEX) (
    CK_VOID_PTR pMutex
);
```

Аргумент функції CK_LOCKMUTEX є покажчиком на м'ютекс-об'єкт, який повинен бути заблокований. Така функція повинна повертати одне з наступних значень: CKR_OK, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MUTEX_BAD.

CK_UNLOCKMUTEX визначається таким чином:

```
typedef CK_CALLBACK_FUNCTION (CK_RV, CK_UNLOCKMUTEX) (
    CK_VOID_PTR pMutex
);
```

Аргумент функції CK_UNLOCKMUTEX є покажчиком на м'ютекс-об'єкт, що буде розблоковано. Така функція повинна повертати одне з наступних значень: CKR_OK, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MUTEX_BAD, CKR_MUTEX_NOT_LOCKED.

Cryptoki підтримує аргументи ініціалізації:

◆ CK_C_INITIALIZE_ARGS; CK_C_INITIALIZE_ARGS_PTR

CK_C_INITIALIZE_ARGS є структурою, що містить необов'язкові аргументи функції C_Initialize.

CK_C_INITIALIZE_ARGS визначається таким чином:

```
typedef struct CK_C_INITIALIZE_ARGS {
    CK_CREATEMUTEX CreateMutex;
    CK_DESTROYMUTEX DestroyMutex;
    CK_LOCKMUTEX LockMutex;
```

```

CK_UNLOCKMUTEX UnlockMutex;
CK_FLAGS flags;
CK_VOID_PTR pReserved;
} CK_C_INITIALIZE_ARGS;

```

Поля структури мають такі значення:

CreateMutex - покажчик на функцію для використання при створенні *mutex*- об'єктів;

DestroyMutex - покажчик на функцію для використання при знищенні *mutex*- об'єктів;

LockMutex - покажчик на функцію для використання при блокуванні *mutex*- об'єктів;

UnlockMutex - покажчик на функцію для використання при розблокуванні *mutex*- об'єктів;

Flags - біт прапорів, що визначають параметри для *C_Initialize*; прапори визначені нижче.

pReserved - зарезервована для використання в майбутньому. Повинно бути *NULL_PTR* для цієї версії *Cryptoki*.

Таблиця 2.11 описує поля прапорів:

Таблиця 2.11. Параметри прапорів *C_Initialize*

Біт прапора	Маска	Опис
CKF_LIBRARY_CANT_CREATE_OS_THREADS	00x00000001	True, якщо програмні потоки, які виконують виклики до бібліотеки, можуть не використовувати звернення, вбудовані (native) до операційної системи, для відтворення нових потоків; false, якщо вони це можуть робити
CKF_OS_LOCKING_OK	00x00000002	True, якщо бібліотека може використовувати вбудовану потокову модель операційної системи для блокування; false, інакше

CK_C_INITIALIZE_ARGS_PTR це покажчик на *CK_C_INITIALIZE_ARGS*

3. Об'єкти

3.1. Загальні визначення

Cryptoki визнає число класів об'єктів, як це визначено в CK_OBJECT_CLASS типу даних. Об'єкт складається з набору атрибутів, кожен з яких має задане значення. Кожен атрибут, що містить об'єкт, має рівно одне значення.

На наступному малюнку показано Cryptoki об'єкти високого рівня ієрархії та деякі атрибути, які вони підтримують:

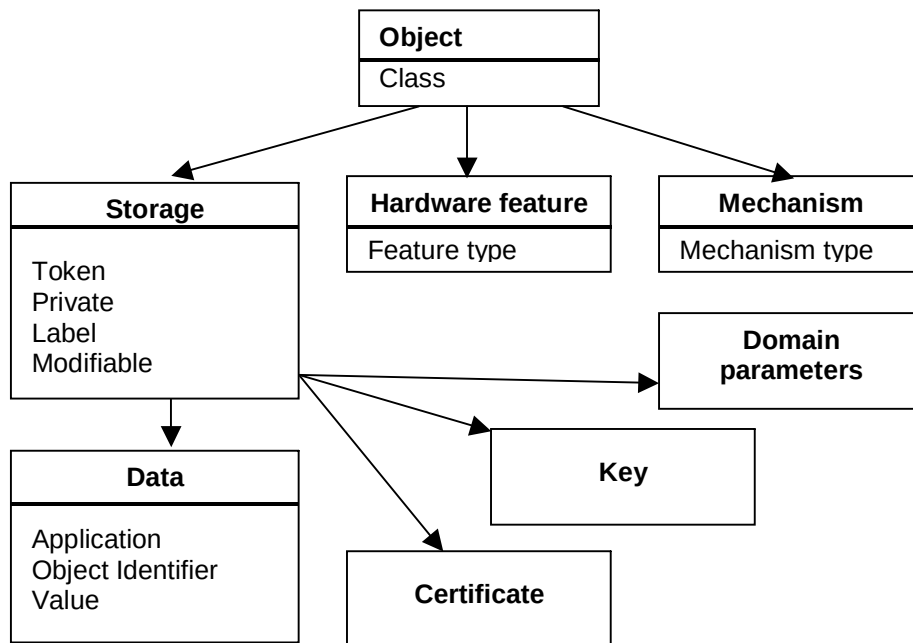


Рис.2.5. Cryptoki об'єкти

Cryptoki надає функції для створення, знищення, копіювання об'єктів в цілому, а також для отримання і зміни значень їх атрибутів. Деякі криптографічні функції (наприклад, C_GenerateKey) також створюють ключові об'єкти для утримання їх як результату.

Об'єкти завжди «правильно сформовані» в Cryptoki, тобто об'єкт завжди містить всі необхідні атрибути і атрибути завжди відповідають один одному з моменту створення об'єкту. Це контрастує з деякими об'єктно-орієнтованими парадигмами, де об'єкт не має атрибутів, і не ініціалізується протягом деякого часу. У Cryptoki, об'єкти завжди ініціалізовані.

Таблиці цього розділу визначають кожен атрибут Cryptoki в термінах типу даних, значення атрибута, і значення атрибута, який може включати за замовчуванням початкове значення. Деякі типи даних визначаються явно Cryptoki (наприклад, CK_OBJECT_CLASS). Значення атрибутів можуть утримувати такі типи:

Big integer – рядок CK_BYTE представляє ціле без знакове число довільного розміру, у форматі Big-Endian (MSB - most significant byte first, від старшого до молодшого, старший байт зберігається першим), тобто найбільш значущий байт перший. Наприклад, ціле число 32768 представлено у вигляді 2-байтового рядка 0x80 0x00;

Local string – не доповнений (unpadded) рядок CK_CHAR (див. Таблиця 2.1) без будь-якого нуль-закінчення (null-termination);

RFC2279string – не доповнений (unpadded) рядок CK_UTF8CHAR без будь-якого нуль-закінчення (null-termination).

Токен може містити кілька ідентичних об'єктів, тобто є припустимим, що два або більше об'єктів мають точно такі ж значення для всіх їх атрибутів.

У більшості випадків кожний тип об'єкта в специфікації Cryptoki володіє абсолютно чітко визначеним набором атрибутів Cryptoki. Деякі з цих атрибутів мають значення за замовчуванням, і не повинні бути вказані при створенні об'єкта, а деякі з цих значень за замовчуванням можуть бути навіть порожнім рядком (""). Тим не менш, об'єкт володіє цими атрибутами. Заданий об'єкт має одне значення для кожного атрибута, яким володіє, навіть якщо атрибутом є атрибут виробника, значення якого виходить за рамки Cryptoki.

На додаток до володіння Cryptoki атрибутами, об'єкти можуть мати додаткові атрибути виробника, смисли і значення яких не передбачено Cryptoki.

3.2. Створення, редагування та копіювання об'єктів

Всі Cryptoki функції, які створюють, змінюють або копіюють об'єкти користуються шаблоном, як одним з аргументів, де шаблон визначає значення атрибутів. Криптографічні функції, які створюють об'єкти, також можуть вносити деякі додаткові значення атрибутів, що залежать від криптографічного механізму, який виконується. У будь-якому випадку, всі необхідні атрибути, що підтримуються об'єктом класу, які не мають значення за замовчуванням, повинні бути вказані при створенні об'єкта чи в шаблоні, чи самої функції.

3.2.1. Створення об'єктів

Об'єкти можуть бути створені за допомогою функції Cryptoki C_CreateObject, C_GenerateKey, C_GenerateKeyPair, C_UnwrapKey і C_DeriveKey, а також копіювання існуючого об'єкта з функцією C_CopyObject (розглядаються в окремому розділі).

Спроба створити об'єкт з будь-якою з цих функцій вимагає відповідного шаблону, який поставляється.

1. Якщо наданий шаблон визначає неприпустимий атрибут, то спроба повинна завершитися з кодом помилки CKR_ATTRIBUTE_TYPE_INVALID. Атрибут є припустимим, якщо він є або одним з атрибутів, описаних у

специфікації Cryptoki, або додатковим атрибутом виробника, що підтримується бібліотекою та токеном.

2. Якщо наданий шаблон визначає неприпустиме значення для дійсного/припустимого атрибута, то спроба повинна завершитися з кодом помилки `SKR_ATTRIBUTE_VALUE_INVALID`. Допустимі значення для атрибутів Cryptoki описані в специфікації Cryptoki.

3. Якщо наданий шаблон визначає значення атрибуту, який є тільки для читання, то спроба повинна завершитися з кодом помилки `SKR_ATTRIBUTE_READ_ONLY`. Є чи ні Cryptoki атрибут тільки для читання явно зазначено в специфікації Cryptoki, однак конкретна бібліотека і токен можуть бути навіть більш суворими, ніж вказується у Cryptoki. Іншими словами, атрибут, який згідно з Cryptoki не тільки для читання, тим не менш може бути тільки для читання при певних обставинах (наприклад, в поєднанні з деякими комбінаціями інших атрибутів) для конкретної бібліотеки і токена. Є чи ні Cryptoki атрибут тільки для читання, очевидно, виходить за рамки Cryptoki.

4. Якщо значення атрибутів у наданому шаблоні, разом із значеннями атрибутів за замовчуванням і значеннями будь-яких атрибутів, наданих самою функцією створення об'єкту, є недостатньо, щоб повністю визначити об'єкт для створення, то спроба повинна завершитися з кодом помилки `SKR_TEMPLATE_INCOMPLETE`.

5. Якщо значення атрибутів у наданому шаблоні, разом із значеннями атрибутів за замовчуванням і значеннями будь-яких атрибутів, наданих самою функцією створення об'єкту, є несумісні, то спроба повинна завершитися з кодом помилки `SKR_TEMPLATE_INCONSISTENT`. Набір значень атрибутів є суперечливим, якщо не всі її члени можуть задовольняти токен одночасно, хоча кожне значення індивідуально діє в Cryptoki. Одним із прикладів несумісного/невідповідного шаблону буде використати шаблон, який визначає два різних значення того ж атрибуту. Іншим прикладом може бути спроба створити секретний ключ, як об'єкт з атрибутом, який підходить для різних видів відкритих особистих ключів, але не для секретних ключів.

6. Якщо наданий шаблон визначає одне і те ж значення для конкретного атрибута більше одного разу (або шаблон визначає, і те ж значення для конкретного атрибута об'єкта міститься у функції створення об'єкта), то поведінка Cryptoki не повністю визначені. Спроба створити об'єкт може бути або успішною, створюючи тим самим той же об'єкт, який був би створений, якщо багато разів заданий атрибут тільки з'явився колись один раз, або це може потерпіти невдачу з кодом помилки `SKR_TEMPLATE_INCONSISTENT`. Розробникам бібліотеки рекомендується зробити у своїх бібліотеках таку поведінку, наче атрибут з'явився тільки один раз на шаблон; розробникам програм застосувань настійно рекомендується ніколи не поставити конкретний атрибут більше одного разу.

Якщо більш ніж одна із ситуацій, зазначених вище, має місце при створенні об'єкту, то кодом помилки, який повертається, може бути будь-який з кодів помилок вище, що має місце.

3.2.2. Модифікація об'єктів

Об'єкти можуть бути змінені за допомогою функції Cryptoki C_SetAttributeValue. Шаблон, що поставляється для C_SetAttributeValue, може містити нові значення для атрибутів, які об'єкт уже має, значення для атрибутів, якими об'єкт ще не володіє, або те й інше.

Деякі атрибути об'єкта можуть бути змінені після створення об'єкту, а деякі не можуть. Крім того, атрибути, які відповідно до Cryptoki піддаються зміні, насправді можуть не бути такими, що модифікуються на певних токенах. Тобто, якщо атрибут Cryptoki описується як такий, що модифікується, то це означає лише, що він може бути змінений. Але остаточно це залежить, зокрема, від токена маркера, який не може насправді підтримувати внесення змін до деяких таких ознаки. Крім того, це може залежати від значень деяких атрибутів об'єкта. Наприклад, CKA_SENSITIVE атрибут секретного ключа може бути змінено з CK_FALSE до CK_TRUE, але не навпаки.

Всі сценарії в розділі 3.2.1 та коди помилок поширюються на модифікацію об'єктів з C_SetAttributeValue, за винятком можливості шаблону бути неповним.

3.2.3. Копіювання об'єктів

Об'єкти можуть бути скопійовані з функцією Cryptoki C_CopyObject. В процесі копіювання об'єкта, C_CopyObject також змінює атрибути новоствореної копії згідно з заданим шаблоном.

Cryptoki атрибути, які можуть бути змінені в ході операції C_CopyObject такі ж, як атрибути Cryptoki, які описані як змінні, а також три спеціальних CKA_TOKEN атрибути, CKA_PRIVATE і CKA_MODIFIABLE. Щоб бути більш точним, ці атрибути можуть модифікуватися в ході C_CopyObject операції. Зокрема токен не може насправді підтримувати внесення змін до деяких таких атрибутів в ході операції C_CopyObject. Крім того, чи може конкретний атрибут об'єкта змінюватися в ході операції C_CopyObject, може залежати від значень деяких атрибутів об'єкта. Наприклад, CKA_SENSITIVE атрибут секретного ключа може бути змінено з CK_FALSE до CK_TRUE, але не навпаки.

Всі сценарії в розділі 3.2.1 та коди помилок поширюються на модифікацію об'єктів з C_SetAttributeValue, за винятком можливості шаблону бути неповним.

3.3. Загальні атрибути об'єктів

Таблиця 3.1. Загальні примітки для таблиць атрибутів об'єкта

Номер примітки	Опис примітки
1.	повинно бути вказано при створенні об'єкта з C_CreateObject
2.	не повинно бути вказано при створенні об'єкта з C_CreateObject
3.	повинно бути вказано, коли об'єкт створюється з C_GenerateKey або C_GenerateKeyPair
4.	не повинно бути вказано, коли об'єкт створюється з C_GenerateKey

	або C_GenerateKeyPair
5.	повинно бути зазначено, коли об'єкт «розгортається»/розшифровується з C_UnwrapKey
6.	не повинно бути вказано, коли об'єкт «розгортається»/розшифровується з C_UnwrapKey
7.	не може бути розкрито, якщо об'єкт має CKA_SENSITIVE атрибут встановлений в CK_TRUE або його CKA_EXTRACTABLE атрибут встановлений в CK_FALSE
8.	може бути змінений після створення об'єкту за допомогою виклику C_SetAttributeValue, або в процесі копіювання об'єктів викликом C_CopyObject. Тим не менш, не виключено, що певний токен не може дозволити зміну атрибутів в ході виклику C_CopyObject
9.	значення за замовчуванням є токен-специфіковане, і може залежати від значень інших атрибутів
10.	може бути встановлений тільки на CK_TRUE SO користувачем
11.	атрибути не можуть бути змінені після встановлення CK_TRUE. Атрибут стає атрибутом тільки для читання.
12.	атрибути не можуть бути змінені після встановлення CK_FALSE. Атрибут стає атрибутом тільки для читання

Таблиця 3.2. Загальні атрибути об'єктів

Атрибут	Тип даних	Значення
CKA_CLASS ¹	CK_OBJECT_CLASS	Клас (тип) об'єкта

Примітки див. у Таблиці 3.1.

Наведені вище таблиці визначають атрибути, спільні для всіх об'єктів.

3.4. Об'єкти апаратних властивостей

Цей розділ визначає об'єкти класу CKO_HW_FEATURE для типу CK_OBJECT_CLASS, як використовується в CKA_CLASS атрибутах об'єктів.

3.4.1. Огляд

Об'єкти апаратних властивостей (CKO_HW_FEATURE) представляють властивості пристрою. Вони забезпечують легко розширюваний спосіб введення нового значення властивості, що базується на інтерфейсі Cryptoki.

Під час пошуку об'єктів за допомогою C_FindObjectsInit і C_FindObjects об'єкти апаратних властивостей, не повертаються, поки CKA_CLASS атрибут в шаблоні не має значення CKO_HW_FEATURE. Це захищає програми застосування, написані для попередніх версій Cryptoki від пошуку об'єктів, які вони не розуміють.

Загальні атрибути об'єктів апаратних властивостей наведено у таблиці:

Таблиця 3.3. Апаратні властивості. Загальні атрибути

Атрибут	Тип даних	Значення
СКА_HW_FEATURE_TYPE ¹	СК_HW_FEATURE	Апаратні властивості (тип)

Примітки див. у таблиці 3.1.

3.4.2. Об'єкт «Годинник»

Атрибут СКА_HW_FEATURE_TYPE приймає значення СКН_CLOCK типу СК_HW_FEATURE.

Об'єкти «Годинник» представляють годинники реального часу, які існують у пристрої. Це означає те ж джерело синхронізації, як utcTime поле структури СК_TOKEN_INFO.

Таблиця 3.4. Атрибути об'єкта «Годинник»

Атрибут	Тип даних	Значення
СКА_VALUE	СК_CHAR [16]	Поточний час як символьний рядок довжиною 16, представлений у форматі YYYYMMDDhhmmssxx (4 символи – рік, 2 символи – місяць, день, година, хвилина та секунда, і 2 додаткові зарезервовані '0' символи).

Атрибут СКА_VALUE може бути встановлено за допомогою функції C_SetAttributeValue, якщо це допускається пристроєм. Сесія, що використовується для встановлення часу, повинна пройти аутентифікацію. Пристрій може вимагати SO для зміни значення часу. C_SetAttributeValue поверне помилку CKR_USER_NOT_LOGGED_IN, щоб показати, що потрібно задати значення іншого типу користувача.

3.4.3. Об'єкт «Монотонний лічильник»

Атрибут СКА_HW_FEATURE_TYPE приймає значення СКН_MONOTONIC_COUNTER типу СК_HW_FEATURE.

Об'єкти «Монотонний лічильник» представляють апаратні лічильники, які існують у пристрої. Лічильник гарантує більшувати кожного разу його значення, що зчитується, але необов'язково по одному. Це може бути використано програмою застосування для генерації серійних номерів, щоб отримати певні гарантії унікальності на токени.

Таблиця 3.5. Атрибути об'єкта «Монотонний лічильник»

Атрибут	Тип даних	Значення
СКА_RESET_ON_INIT ¹	CK_BBOOL	Значення лічильника повернеться до раніше поверненого значення, якщо токен ініціалізується за допомогою C_InitializeToken.
СКА_HAS_RESET ¹	CK_BBOOL	Значення лічильника було скинуто/очищено не рідше одного разу на певний момент часу
СКА_VALUE ¹	Byte array	Поточна версія монотонного лічильника. Значення повертається у Big Endian форматі

¹ Тільки для читання

Атрибут СКА_VALUE не може бути встановлено за бажанням клієнта.

3.4.4. Об'єкти інтерфейсу користувача

Атрибут СКА_HW_FEATURE_TYPE приймає значення SKH_USER_INTERFACE типу CK_HW_FEATURE.

Об'єкти інтерфейсу користувача представляють презентацію можливостей пристрою.

Таблиця 3.6. Атрибути об'єкта «Інтерфейс користувача»

Атрибут	Тип даних	Значення
СКА_PIXEL_X	CK_ULONG	Роздільна здатність екрана (в пікселях) по X-осі (наприклад, 1280)
СКА_PIXEL_Y	CK_ULONG	Роздільна здатність екрана (в пікселях) по Y-осі (наприклад, 1024)
СКА_RESOLUTION	CK_ULONG	DPI, крапок на дюйм
СКА_CHAR_ROWS	CK_ULONG	Для символно-орієнтованих дисплеїв, кількість символних рядків (наприклад, 24)
СКА_CHAR_COLUMNS	CK_ULONG	Для символно-орієнтованих дисплеїв: кількість символних стовпців (наприклад, 80). Якщо дисплей типу пропорційного шрифту, то це ширина дисплея в "em" (буква "M"), див CC / PP структури.

Атрибут	Тип даних	Значення
CKA_COLOR	CK_BBOOL	Підтримка кольорів
CKA_BITS_PER_PIXEL	CK_ULONG	число біт кольорів або відтінки сірого на піксель
CKA_CHAR_SETS	RFC 2279 рядок	Рядок, який вказує набори символів, що підтримуються, як визначено наборами IANA MIBenum (www.iana.org). Підтримувані набори символів, розділені ";". Наприклад, знак підтримки ISO-8859-1 і US-ASCII буде встановлювати значення атрибута "4; 3"
CKA_ENCODING_METHODS	RFC 2279 рядок	Рядок, який вказує методи кодування передачі контенту, що підтримуються, як це визначено в IANA (www.iana.org). Підтримувані методи розділяються символом ";". Наприклад, токен що підтримує 7bit, 8bit і base64, може встановити значення атрибута "7bit; 8bit, base64".
CKA_MIME_TYPES	RFC 2279 рядок	Рядок, який вказує підтримувані (презентабельно) MIME-типи, як це визначено в IANA (www.iana.org). Підтримувані типи розділяються символом ";". Наприклад, токен, що підтримує MIME типи "/" б", "А / С" і "а / д", буде встановлювати значення атрибута "/" б; / с, а / д".

Вибір атрибутів, і пов'язані з ними типи даних, було зроблено з метою залишитися у відповідності до RFC 2534 і CC/PP Struct, наскільки це можливо. Спеціальне CK_UNAVAILABLE_INFORMATION значення може бути використане для атрибутів, що базуються на CK_ULONG, якщо інформація недоступна або непридатна.

Жодне із значень атрибутів може бути не встановлено.

Значення атрибута `CKA_ENCODING_METHODS` можуть бути використані, коли застосуванню необхідно відправляти до токена MIME-об'єкти з закодованими змістом.

3.5. Об'єкти зберігання

Об'єкти зберігання (Storage Objects) - це не клас об'єкта, а отже, ніякого `CKO_` визначення не потрібно. Це категорія класів об'єктів із загальними атрибутами для класів об'єктів.

Таблиця 3.7. Загальні атрибути об'єктів зберігання

Атрибут	Тип даних	Значення
<code>CKA_TOKEN</code>	<code>CK_BBOOL</code>	<code>CK_TRUE</code> , якщо об'єкт є об'єктом токена; <code>CK_FALSE</code> якщо об'єкт є сесійним. За замовчуванням є <code>CK_FALSE</code> .
<code>CKA_PRIVATE</code>	<code>CK_BBOOL</code>	<code>CK_TRUE</code> , якщо об'єкт є особистим об'єктом; <code>CK_FALSE</code> , якщо об'єкт є відкритим об'єктом. Значення за замовчуванням є токен-специфіковані, і може залежати від значень інших атрибутів об'єкта
<code>CKA_MODIFIABLE</code>	<code>CK_BBOOL</code>	<code>CK_TRUE</code> , якщо об'єкт може бути змінений; за замовчуванням <code>CK_TRUE</code> .
<code>CKA_LABEL</code>	RFC2279 рядок	Опис об'єкту (за замовчуванням пусто).

Тільки атрибут `CKA_LABEL` може бути змінений після створення об'єкту. `CKA_TOKEN`, `CKA_PRIVATE` і `CKA_MODIFIABLE` атрибути можуть бути змінені тільки в процесі копіювання об'єкта.

`CKA_TOKEN` атрибут визначає, чи є об'єкт об'єктом токена, чи об'єктом сесії.

Коли `CKA_PRIVATE` атрибут є `CK_TRUE`, то користувач не може отримати доступ до об'єкта, поки користувач не пройшов аутентифікацію у токені.

Значення атрибута `CKA_MODIFIABLE` визначає, чи дійсно об'єкт тільки для читання. Він може (`CK_TRUE`) або не може бути таким, що є дозволяє редагування та видалення.

Атрибут `CKA_LABEL` призначений для допомоги користувачам під час перегляду (використовується як «дружнє ім'я»).

3.6. Об'єкти «Дані»

Об'єкт «Дані» (Data object) класу CKO_DATA типу CK_OBJECT_CLASS, як використовується в CKA_CLASS атрибутах об'єктів, містить відомості, встановлені програмою застосування. Крім забезпечення доступу до них, Cryptoki не надає особливого значення для об'єкта даних.

У таблиці нижче зазначено атрибути, що підтримуються об'єктами даних, на додаток до загальних властивостей, визначених для даного класу об'єктів:

Таблиця 3.8. Атрибути об'єкту «Дані»

Атрибут	Тип даних	Значення
CKA_APPLICATION	RFC2279 рядок	Опис програми застосування, що керує об'єктом (за замовчуванням пусто)
CKA_OBJECT_ID	Byte Array	DER-кодований ідентифікатор об'єкта із зазначенням типу об'єкта даних (за замовчуванням пусто)
CKA_VALUE	Byte Array	Байт масив об'єкта (за замовчуванням пусто)

Атрибут CKA_APPLICATION надає засоби для програм застосування, призначені вказати на власника об'єкта даних, який ними керує. Однак, Cryptoki не гарантує, що тільки конкретна програма застосування має доступ до об'єкта даних.

CKA_OBJECT_ID атрибут надає програмам незалежний і розширюваний спосіб вказати тип значення об'єкта даних. Cryptoki не передбачає способи гарантій того, що ідентифікатор об'єкта даних відповідає значенню даних.

Нижче наведено приклад шаблону, що містить атрибути для створення об'єкта даних:

```
CK_OBJECT_CLASS class = CKO_DATA;
CK_UTF8CHAR label[] = "A data object";
CK_UTF8CHAR application[] = "An application";
CK_BYTE data[] = "Sample data";
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_APPLICATION, application, sizeof(application)-1},
    {CKA_VALUE, data, sizeof(data)}
};
```

3.7. Об'єкти «Сертифікат»

3.7.1. Загальні атрибути

Об'єкти «Сертифікат» (об'єкт класу CKO_CERTIFICATE) утримують відкритий ключ або атрибути сертифіката. Крім забезпечення доступу до об'єктів «Сертифікат», Cryptoki не надає особливого сенсу сертифікатам.

Таблиця нижче визначає загальні атрибути об'єкта «Сертифікат», на додаток до загальних властивостей, визначених для даного класу об'єктів:

Таблиця 3.9. Загальні атрибути об'єктів «Сертифікат»

Атрибут	Тип даних	Значення
CKA_CERTIFICATE_TYPE ¹	CK_CERTIFICATE_TYPE	Тип сертифікат
CKA_TRUSTED ¹⁰	CK_BBOOL	Сертифікат може бути довіреними для програми, якою він був створений.
CKA_CERTIFICATE_CATEGORY	CK_ULONG	Класифікація сертифікатів: 0= не визначено (значення за замовчуванням), 1= користувача (token user), 2= центра сертифікації (authority), 3 = інші особи (other entity)
CKA_CHECK_VALUE	Byte array	Контрольна сума
CKA_START_DATE	CK_DATE	Дату початку чинності сертифікату (за замовчуванням пусто)
CKA_END_DATE	CK_DATE	Дата закінчення дії сертифікату (за замовчуванням пусто)

Примітки див. у Таблиці 3.1.

Атрибут CKA_CERTIFICATE_TYPE не може бути змінений після створення об'єкту. Ця версія Cryptoki підтримує такі типи сертифікатів:

X.509 public key certificate (X.509 сертифікат відкритого ключа);

WTLS public key certificate (WTLS сертифікат відкритого ключа);
X.509 attribute certificate (X.509 сертифікат атрибутів).

Атрибут `СКА_CERTIFICATE_CATEGORY` використовується для того, щоб вказати, якщо сертифікат, що зберігається, є сертифікатом користувача, для якого відповідний особистий ключ можна отримати на токени («token user»), сертифікат центру сертифікації («authority»), або інших кінцевих користувачів сертифіката («other entity»). Цей атрибут не може бути змінений після створення об'єкту.

`СКА_CERTIFICATE_CATEGORY` і `СКА_TRUSTED` атрибути будуть спільно використовуватися для класифікації сертифікатів.

`СКА_CHECK_VALUE`: значення цього атрибута походить від сертифікату, узявши перші три байти SHA-1 хеш від `СКА_VALUE` атрибута об'єкта сертифікат.

`СКА_START_DATE` і `СКА_END_DATE` атрибути призначені тільки для довідки; Cryptoki не надає особливого значення цим атрибутам. Якщо вони присутні, програма застосування відповідає за встановлення їх значення, відповідно до полів сертифіката.

3.7.2. Об'єкти «X.509 сертифікат» відкритого ключа

Об'єкти «X.509 сертифікат» (сертифікат типу `СКС_X_509`) утримують X.509 сертифікати відкритих ключів. Таблиця нижче визначає атрибути об'єкта «X.509 сертифікат», на додаток до загальних властивостей, визначених для даного класу об'єктів:

Таблиця 3.10. Атрибути об'єкта «X.509 сертифікат»

Атрибут	Тип даних	Значення
<code>СКА_SUBJECT</code> ¹	Byte array	DER-кодоване ім'я суб'єкта сертифікату
<code>СКА_ID</code>	Byte array	Ідентифікатор ключа для відкритих/особистих ключів (за замовчуванням пусто)
<code>СКА_ISSUER</code>	Byte array	DER-кодоване ім'я емітента сертифікату (за замовчуванням пусто)
<code>СКА_SERIAL_NUMBER</code>	Byte array	DER-кодований серійний номер сертифікату (за замовчуванням пусто)

Атрибут	Тип даних	Значення
CKA_VALUE ²	Byte array	BER-кодований сертифікат
CKA_URL ³	RFC2279 рядок	Якщо не порожній, то цей атрибут дає адресу, де може бути отриманий повний сертифікат (за замовчуванням пусто)
CKA_HASH_OF_SUBJECT_PUBLIC_KEY ⁴	Byte array	SHA-1 хеш відкритого ключа суб'єкта (за умовчанням пусто)
CKA_HASH_OF_ISSUER_PUBLIC_KEY ⁴	Byte array	SHA-1 хеш відкритого ключа емітента (за умовчанням пусто)
CKA_JAVA_MIDP_SECURITY_DOMAIN	CK_ULONG	Java MIDP домен безпеки: 0= не визначено (значення за замовчуванням), 1= виробника (manufacturer), 2= оператор (operator), 3= третя сторона (third party)

¹ Повинні бути вказані при створенні об'єкта.

² Повинні бути вказані при створенні об'єкта. Повинно бути не порожнім, якщо CKA_URL порожній.

³ Повинні бути не порожнім, якщо CKA_VALUE порожній.

⁴ Може бути порожнім, тільки якщо CKA_URL порожній.

Тільки CKA_ID, CKA_ISSUER і CKA_SERIAL_NUMBER атрибути можуть бути змінені після створення об'єкту.

CKA_ID атрибут призначений як засіб розрізнення декількох пар особистий-ключ/секретний-ключ, що мають однакове значення імені суб'єкту (чи зберігається в тому ж токени чи ні). Він призначений в інтересах взаємодії для того, щоб ім'я суб'єкта та ідентифікатор ключа сертифіката були такими ж, як і для відповідних відкритих і особистих ключів. Тим не менш, Cryptoki не нав'язує

цю асоціацію, і навіть унікальності ідентифікатора ключа для даного об'єкта, зокрема, програма може залишити ідентифікатор ключа порожнім.

СКА_ISSUER і СКА_SERIAL_NUMBER атрибути для сумісності з PKCS#7 і Privacy Enhanced Mail (RFC1421). Зверніть увагу, що з 3-ї версії розширення для сертифікатів X.509, ідентифікатор ключа може зберігатися в сертифікаті. Передбачається, що значення СКА_ID буде ідентичним ідентифікатору ключа в такому розширенні сертифіката, хоча це не обов'язково по Cryptoki.

СКА_URL атрибут дозволяє підтримувати для зберігання URL, де може бути знайдений сам сертифікат. Зберігання URL замість повного сертифіката часто використовується в мобільних середовищах.

СКА_HASH_OF_SUBJECT_PUBLIC_KEY і СКА_HASH_OF_ISSUER_PUBLIC_KEY атрибути використовуються для зберігання хеш-значення відкритих ключів суб'єкта та емітента. Вони особливо важливі, коли доступно тільки URL, щоб мати можливість співвідносити сертифікат із особистим ключем і під час пошуку сертифіката емітента.

СКА_JAVA_MIDP_SECURITY_DOMAIN атрибут пов'язує сертифікат з доменом безпеки Java MIDP.

Нижче наведено приклад шаблону для створення об'єкту X.509 сертифікат:

```
CK_OBJECT_CLASS class = CKO_CERTIFICATE;
CK_CERTIFICATE_TYPE certType = CKC_X_509;
CK_UTF8CHAR label[] = "A certificate object";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE certificate[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_CERTIFICATE_TYPE, &certType, sizeof(certType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_SUBJECT, subject, sizeof(subject)},
    {CKA_ID, id, sizeof(id)},
    {CKA_VALUE, certificate, sizeof(certificate)}
};
```

3.7.3. Об'єкти «WTLS сертифікат» відкритого ключа

Об'єкти «WTLS сертифікат» відкритого ключа (типу CKC_WTLS сертифікат) утримує відкритий ключ WTLS сертифікат (WTLS, Wireless Transport Layer Security – Протокол безпеки бездротового транспортного рівня; TLS, Transport Layer Security - Протокол безпеки транспортного рівня, заснований на SSL 3.0 і повинен прийти йому на заміну, описано в RFC 2246).

У таблиці нижче наведено атрибути об'єктів «WTLS сертифікат», на додаток до загальних властивостей, визначених для даного класу об'єктів.

Таблиця 3.11. Атрибути об'єкта «WTLS сертифікат»

Атрибут	Тип даних	Значення
СКА_SUBJECT ¹	Byte array	WTLS-кодований (тип Ідентифікатор) суб'єкт сертифікату.
СКА_ISSUER	Byte array	WTLS-кодований (тип Ідентифікатор) емітент сертифікату (за замовчуванням пусто)
СКА_VALUE ²	Byte array	WTLS-кодований сертифікат
СКА_URL ³	RFC2279 рядок	Якщо не порожній, то цей атрибут дає адресу, де може бути отриманий повний сертифікат
СКА_HASH_OF_SUBJECT_PUBLIC_KEY ⁴	Byte array	SHA-1 хеш відкритого ключа суб'єкта (за умовчанням пусто)
СКА_HASH_OF_ISSUER_PUBLIC_KEY ⁴	Byte array	SHA-1 хеш відкритого ключа емітента (за замовчуванням пусто)

¹ Повинні бути вказані при створенні об'єкта. Може бути порожнім, якщо СКА_VALUE порожній.

² Повинені бути вказані при створенні об'єкта. Повинно бути не порожнім, якщо СКА_URL порожній.

³ Повинно бути не порожнім, якщо СКА_VALUE порожній.

⁴ Може бути тільки порожнім, якщо СКА_URL порожній.

Тільки СКА_ISSUER атрибут може бути змінений після створення об'єкту.

Кодування для СКА_SUBJECT, СКА_ISSUER і СКА_VALUE атрибутів можна знайти в «WAP. Wireless Transport Layer Security Version — WAP-261-WTLS-20010406-a. April 2001. URL: <http://www.wapforum.org/>».

СКА_URL атрибут дозволяє утримувати для URL, де може бути знайдений сам сертифікат. Зберігання URL замість повного сертифіката часто використовується в мобільних середовищах.

СКА_HASH_OF_SUBJECT_PUBLIC_KEY і СКА_HASH_OF_ISSUER_PUBLIC_KEY атрибути використовуються для зберігання хеш відкритих ключів суб'єкта і емітента. Вони особливо важливі, коли тільки URL доступна, щоб мати можливість співвідносити сертифікат із закритим ключем і під час пошуку сертифіката емітента.

Нижче наведено приклад шаблону для створення об'єкта «WTLS сертифікат»:

```
CK_OBJECT_CLASS class = CKO_CERTIFICATE;
CK_CERTIFICATE_TYPE certType = CKC_WTLS;
CK_UTF8CHAR label[] = "A certificate object";
CK_BYTE subject[] = {...};
CK_BYTE certificate[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] =
{
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_CERTIFICATE_TYPE, &certType, sizeof(certType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_SUBJECT, subject, sizeof(subject)},
    {CKA_VALUE, certificate, sizeof(certificate)}
};
```

3.7.4. Об'єкти «X.509 сертифікат атрибутів»

Об'єкти «X.509 сертифікат атрибутів» (сертифікат типу CKC_X_509_ATTR_CERT) утримують сертифікати X.509 сертифікат атрибутів.

Таблиця нижче визначає атрибути об'єкта «X.509 сертифікат атрибутів», на додаток до загальних властивостей, визначених для даного класу об'єктів:

Таблиця 3.12. Атрибути об'єкта «X.509 сертифікат атрибутів»

Атрибут	Тип даних	Значення
СКА_OWNER ¹	Byte array	DER-кодоване поле суб'єкта сертифіката. Це відрізняється від СКА_SUBJECT атрибута, що міститься в СКА_X_509 сертифікатах, тому що ASN.1 синтаксис та кодування різні.
СКА_AC_ISSUER	Byte array	DER-кодоване поле емітента сертифіката. Це відрізняється від

Атрибут	Тип даних	Значення
		СКА_ISSUER атрибута, що міститься в СКС_X_509 сертифікатах, тому ASN.1 синтаксис та кодування різні. (За замовчуванням пусто)
СКА_SERIAL_NUMBER	Byte array	DER-кодований серійний номер сертифікату (За замовчуванням пусто)
СКА_ATTR_TYPES	Byte array	BER-кодована послідовність (sequence) значень об'єктних ідентифікаторів відповідних типів атрибутів, що містяться в сертифікаті. Якщо вони присутні, то це поле дає можливість програмі для пошуку конкретного атрибута сертифіката, не витягуючи і не розбираючи (parsing) самого сертифіката. (За замовчуванням пусто)
СКА_VALUE ¹	Byte array	BER-кодований сертифікат.

¹ Повинні бути вказані при створенні об'єкта

Тільки СКА_AC_ISSUER, СКА_SERIAL_NUMBER і СКА_ATTR_TYPES атрибути можуть бути змінені після створення об'єкту.

Нижче наведено приклад шаблону для створення об'єкта «X.509 сертифікат атрибутів»:

```

CK_OBJECT_CLASS class = CKO_CERTIFICATE;
CK_CERTIFICATE_TYPE certType = CKC_X_509_ATTR_CERT;
CK_UTF8CHAR label[] = "An attribute certificate object";
CK_BYTE owner[] = {...};
CK_BYTE certificate[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_CERTIFICATE_TYPE, &certType, sizeof(certType)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_OWNER, owner, sizeof(owner)},
    {CKA_VALUE, certificate, sizeof(certificate)}
};

```

3.7.5. Об'єкти «Ключ»

Немає СКО_ визначення для базового класу об'єкту «Ключі», лише для типів ключа, отриманих з нього.

Цей розділ визначає клас об'єкта СКО_PUBLIC_KEY, СКО_PRIVATE_KEY та СКО_SECRET_KEY для типу СК_OBJECT_CLASS, що використовуються атрибутом об'єктів СКА_CLASS.

Об'єкти «Ключ», що містять ключі шифрування або автентифікації, можуть бути відкритими, особистими або секретними ключами.

Наступні загальні положення застосовуються у всіх таблицях, які описують атрибути ключів.

3.7.6. Загальні атрибути

Таблиця нижче визначає атрибути відкритого, особистого та секретного ключа та загальні атрибути, визначені для цього класу об'єкта:

Таблиця 3.13. Загальні атрибути ключів

Атрибути	Тип даних	Опис
СКА_KEY_TYPE ^{1,5}	СК_KEY_TYPE	Тип ключа
СКА_ID ⁸	Byte array	Ідентифікатор ключа (за замовчуванням може бути порожнім, визначається виробником токена)
СКА_START_DATE ⁸	СК_DATE	Початкова дата ключа (за замовчуванням є порожньою)
СКА_END_DATE ⁸	СК_DATE	Кінцева дата ключа (за замовчуванням є порожньою)
СКА_DERIVE ⁸	СК_BBOOL	СК_TRUE, якщо ключ підтримує виробництво ключа – derivation, тобто, якщо інші ключі можуть бути отримані від цього (за замовченням СК_FALSE)
СКА_LOCAL ^{2,4,6}	СК_BBOOL	СК_TRUE, тільки якщо ключ був або згенеровано локально (тобто, у токені) шляхом виклику C_GenerateKey або C_GenerateKeyPair створено шляхом виклику C_CopyObject, як копія ключа, який мав атрибут СКА_LOCAL, що встановлено у СК_TRUE
СКА_KEY_GEN_MECHANISM ^{2,4,6}	СК_MECHANISM_TYPE	Ідентифікатор механізму, що використовується для генерації ключового матеріалу.
СКА_ALLOWED_MECHANISMS	СК_MECHANISM_TYPE_PTR,	Список механізмів, що можуть бути застосовані цим ключем. Число

Атрибути	Тип даних	Опис
	показчик на CK_MECHANISM_TYPE array	механізмів у масиві є компонентом атрибуту ulValueLen, що ділиться на розмір CK_MECHANISM_TYPE

Примітки див. у Таблиці 3.1.

Поле CKA_ID має призначення розрізнити ключі серед їх множини. У випадку відкритих і особистих ключів, це поле допомагає при обробці множини ключів, що мають однакове значення атрибуту «Суб'єкт» (subject); ідентифікатор відкритого ключа і його відповідного особистого ключа *повинен бути однаковим*.

Ідентифікатор ключа *повинен бути однаковим* також для відповідного сертифіката відкритого ключа, якщо він існує.

У випадку секретних ключів, значення атрибуту CKA_ID залежить від програми застосування.

Відмітьте, що атрибути CKA_START_DATE та CKA_END_DATE застосовуються тільки для посилення; Cryptoki не надає ніякого спеціального значення для них. Зокрема, це не обмежує використання ключа у відповідності з датами; виконання цього стосується програми застосування.

Атрибут CKA_DERIVE має значення CK_TRUE, якщо і тільки якщо можна отримати інші ключі від цього ключа.

Атрибут CKA_LOCAL має значення CK_TRUE, якщо і тільки якщо значення ключа генерувалося у токени шляхом виклику C_GenerateKey або C_GenerateKeyPair.

Атрибут CKA_KEY_GEN_MECHANISM ідентифікує механізм генерації, що застосовується для генерації ключового матеріалу. Він містить валідне значення тільки, якщо атрибут CKA_LOCAL має значення CK_TRUE. Якщо CKA_LOCAL має значення CK_FALSE, то значення атрибуту повинно бути CK_UNAVAILABLE_INFORMATION.

3.7.7. Об'єкти «Відкриті ключі»

Об'єкти класу SKO_PUBLIC_KEY містять відкриті ключі.

Таблиця нижче визначає загальні атрибути для усіх відкритих ключів, на додаток до загальних атрибутів, визначених для цього класу об'єкту:

Таблиця 3.14. Загальні атрибути відкритих ключів

Атрибути	Тип даних	Опис
CKA_SUBJECT ⁸	Byte array	DER-кодування імені суб'єкта ключа (за замовченням є порожнім)
CKA_ENCRYPT ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує шифрування

Атрибути	Тип даних	Опис
CKA_VERIFY ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує верифікацію (перевіряння) підпису, де підпис додається до даних
CKA_VERIFY_RECOVER ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує верифікацію (перевіряння) підпису, де дані повертаються підписом (вилучаються із підпису)
CKA_WRAP ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує «обгорнення», тобто може бути використаний для «обгорнення»/зашифрування іншого ключа
CKA_TRUSTED ¹⁰	CK_BBOOL	Ключ може бути довіреним для програми, якою він був створений. Ключ «обгорнення» може бути використано для «обгорнення» ключів 3 CKA_WRAP_WITH_TRUSTED, що встановлений у CK_TRUE.
CKA_WRAP_TEMPLATE	CK_ATTRIBUTE_PTR	Для ключів «обгорнення». Шаблон атрибутів, для порівняння з будь-якими «обгорненими» ключами, що «обгорнені» з використанням цього ключа «обгорнення». Ключі, які не відповідають, не можуть бути «обгорненими». Число атрибутів у масиві є ulValueLen атрибуту, що ділиться на розмір CK_ATTRIBUTE.

Примітки див. у Таблиці 3.1.

З метою функціональної сумісності це призначається для того, щоб ім'я суб'єкта (subject name) та ідентифікатор ключа для відкритого ключа були тими ж, як і для відповідного сертифіката і особистого ключа. Проте, Cryptoki не вимагає, щоб сертифікат та особистий ключ також зберігалися у токени.

Для узгодження застосування keyUsage прапорів у ISO/IEC 9594-8 (X.509) для відкритих ключів та атрибутів відкритих ключів у PKCS#11, використовується така таблиця:

Таблиця 3.15. Застосування прапорів відкритих ключів

Прапори сертифікатів X. 509	Cryptoki атрибути
dataEncipherment	CKA_ENCRYPT
digitalSignature, keyCertSign, cRLSign	CKA_VERIFY
digitalSignature, keyCertSign, cRLSign	CKA_VERIFY_RECOVER

keyAgreement	CKA_DERIVE
keyEncipherment	CKA_WRAP
nonRepudiation	CKA_VERIFY
nonRepudiation	CKA_VERIFY_RECOVER

3.7.8. Об'єкти «Особисті ключі»

Об'єкти класу CKO_PRIVATE_KEY містять особисті ключі.

Таблиця нижче визначає загальні атрибути для усіх особистих ключів, на додаток до загальних атрибутів, визначених для цього класу об'єкта:

Таблиця 3.16. Загальні атрибути особистих ключів

Атрибути	Тип даних	Опис
CKA_SUBJECT ⁸	Byte array	DER-кодування імені суб'єкта сертифіката (за замовчуванням є порожнім)
CKA_SENSITIVE ^{8,11}	CK_BBOOL	CK_TRUE, якщо ключ є чутливим
CKA_DECRYPT ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує розшифрування
CKA_SIGN ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує підписування, де підпис є додатком до даних
CKA_SIGN_RECOVER ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує підписування, де дані можуть бути відновлені з підпису
CKA_UNWRAP ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує «розгорнення», тобто може бути використаний для «розгорнення»/розшифрування іншого ключа
CKA_EXTRACTABLE ^{8,12}	CK_BBOOL	CK_TRUE, якщо ключ є таким, що вилучається, та може бути «обгорнутим»/зашифрованим
CKA_ALWAYS_SENSITIVE ^{2,4,6}	CK_BBOOL	CK_TRUE, якщо ключ завжди має атрибут CKA_SENSITIVE,

Атрибути	Тип даних	Опис
		встановлений у CK_TRUE
CKA_NEVER_EXTRACTABLE ^{2,4,6}	CK_BBOOL	CK_TRUE, якщо ключ ніколи не має атрибуту CKA_EXTRACTABLE, що встановлений у CK_TRUE
CKA_WRAP_WITH_TRUSTED ¹¹	CK_BBOOL	CK_TRUE, якщо ключ може бути «обгорнутий» тільки за допомогою ключа обгорнення, який має атрибут CKA_TRUSTED, встановлений у CK_TRUE. Значення за замовчуванням - CK_FALSE
CKA_UNWRAP_TEMPLATE	CK_ATTRIBUTE_PTR	Для «обгорнення»/зашифрування ключів. Шаблон атрибутів, що застосовується до будь-яких «обгорнених» ключів, використовуючи цей ключ «обгорнення». Будь-який шаблон користувача, що використовується після цього шаблону, якщо об'єкт вже був створений. Число атрибутів в масиві є ulValueLen, що ділиться на розмір CK_ATTRIBUTE.
CKA_ALWAYS_AUTHENTICATE	CK_BBOOL	Якщо CK_TRUE, то користувачу доводиться використовувати PIN для кожної дії (підписання або розшифрування) з ключем. Значення за замовчуванням - CK_FALSE

Примітки див. у Таблиці 3.1.

З метою функціональної сумісності це призначається для того, щоб ім'я суб'єкта (subject name) та ідентифікатор ключа для особистого ключа були тими ж, як і для відповідного сертифіката і відкритого ключа. Проте, Cryptoki не вимагає, щоб сертифікат та відкритий ключ також зберігалися у токени.

Якщо атрибут CKA_SENSITIVE, або якщо атрибут CKA_EXTRACTABLE мають значення CK_TRUE, то певні атрибути особистого ключа не можуть бути виявлені/ розкриті в звичайному тексті поза межами токена. Які з атрибутів, - це зазначено для кожного типу особистих ключів в таблиці атрибутів в розділі, що описує типи ключів.

Атрибут CKA_ALWAYS_AUTHENTICATE може бути використаний для того, щоб здійснити повторну автентифікацію (тобто змушують користувача забезпечити PIN) для кожного застосування приватного ключа. "Використання" в цьому випадку має на увазі криптографічну операцію таку, як наприклад, підписання або розшифрування. Цей атрибут можна тільки встановити у CK_TRUE, коли CKA_PRIVATE також має значення CK_TRUE.

Повторна аутентифікація відбувається під час виклику функції C_Login, коли userType встановлено у CKU_CONTEXT_SPECIFIC негайно після того, як криптографічна операція, що використовує ключ, була ініціалізована (наприклад після C_SignInit). У цьому виклику, фактичний тип користувача беззастережно надає вимоги використання активного ключа. Якщо C_Login повертає CKR_OK, то користувач був успішно авторизований і це встановлює активний ключ в авторизований стан, доки криптографічна операція не буде успішно або невдало завершеною (наприклад C_Sign, C_SignFinal,.). Повернене значення CKR_PIN_INCORRECT від C_Login означає, що користувачу було відмовлено у використанні ключа та продовженні криптографічної операції, ніби то C_Login не була викликана. У обох цих випадках, стан сесії збережеться, проте повторна спроба аутентифікації буде невдалою, це може примусити заблокувати PIN. C_Login повертає в цьому випадку CKR_PIN_LOCKED і це також призводить до виходу користувача з токена. Терплячи невдачу або опускаючи повторну автентифікацію, коли CKA_ALWAYS_AUTHENTICATE встановлено у CK_TRUE, повинно призводити до повернення CKR_USER_NOT_LOGGED_IN при спробі виконання операцій з використанням ключа.

Коли C_Login поверне CKR_OPERATION_NOT_INITIALIZED при повторній аутентифікації, то це не впливатиме на активну криптографічну операцію, якщо CKA_ALWAYS_AUTHENTICATE встановлено у CK_FALSE.

3.7.9. Об'єкти «Секретні ключі»

Об'єкти класу CKO_SECRET_KEY містять секретні ключі.

Таблиця нижче визначає загальні атрибути усіх секретних ключів, на додаток до загальних атрибутів, визначених для цього класу об'єкту.

Таблиця 3.17. Загальні атрибути секретних ключів

Атрибути	Тип даних	Опис
CKA_SENSITIVE ^{8,11}	CK_BBOOL	CK_TRUE , якщо об'єкт є чутливим (за замовчуванням CK_FALSE)
CKA_ENCRYPT ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує зашифрування
CKA_DECRYPT ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує розшифрування
CKA_SIGN ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує підписування (тобто коди аутентифікації), де підпис – є додатком до даних
CKA_VERIFY ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує верифікацію (перевірку) (тобто коди аутентифікації), де підпис – є додатком до даних
CKA_WRAP ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує «обгорнення», тобто може бути використаний для зашифрування іншого ключа
CKA_UNWRAP ⁸	CK_BBOOL	CK_TRUE, якщо ключ підтримує «розгорнення», тобто може бути використаний для розшифрування іншого ключа
CKA_EXTRACTABLE ^{8,12}	CK_BBOOL	CK_TRUE, якщо ключ є таким, що вилучається, та може бути «обгорнутим»/ зашифрованим
CKA_ALWAYS_SENSITIVE ^{2,4,6}	CK_BBOOL	CK_TRUE, якщо ключ завжди має атрибут CKA_SENSITIVE, встановлений у CK_TRUE
CKA_NEVER_EXTRACTABLE ^{2,4,6}	CK_BBOOL	CK_TRUE, якщо ключ ніколи не має атрибуту CKA_EXTRACTABLE, що

Атрибути	Тип даних	Опис
		встановлений у CK_TRUE
CKA_CHECK_VALUE	Byte array	Контрольна сума ключа
CKA_WRAP_WITH_TRUSTED ¹¹	CK_BBOOL	CK_TRUE, якщо ключ може бути «обгорнутий» тільки за допомогою ключа обгорнення, який має атрибут CKA_TRUSTED, встановлений у CK_TRUE. Значення за замовчуванням - CK_FALSE
CKA_TRUSTED ¹⁰	CK_BBOOL	Ключ «обгорнення» може бути використаний для зашифрування ключів з атрибутом CKA_WRAP_WITH_TRUSTED, встановленим у CK_TRUE.
CKA_UNWRAP_TEMPLATE	CK_ATTRIBUTE_PTR	Для «обгорнення»/зашифрування ключів. Шаблон атрибутів, що застосовується до будь-яких «обгорнених» ключів, використовуючи цей ключ «обгорнення». Будь-який шаблон користувача, що використовується після цього шаблону, якщо об'єкт вже був створений. Число атрибутів в масиві є ulValueLen, що ділиться на розмір CK_ATTRIBUTE.
CKA_ALWAYS_AUTHENTICATE	CK_BBOOL	Якщо CK_TRUE, то користувачу доводиться використовувати PIN для кожної дії (підписання або розшифрування) з ключем. Значення за замовчуванням - CK_FALSE

Примітки див. у Таблиці 3.1.

Якщо атрибут CKA_SENSITIVE є CK_TRUE, або якщо атрибут CKA_EXTRACTABLE є CK_FALSE, то певні атрибути секретного ключа не

можуть бути виявлені/ розкриті в звичайному тексті поза межами точену. Які це атрибути, вказано для кожного типу секретного ключа в таблиці атрибутів в розділі, що описує той тип ключа.

Атрибут «Контрольне значення ключа» (KCV, key check value) для симетричних ключів називається `СКА_CHECK_VALUE`, тип масив байтів, довжина 3 байти, діє подібно до відбитку пальця, або контрольної суми ключа. Атрибут повинен бути використаний для перехресної перевірки (cross-check) симетричних ключів проти інших систем, де такі ж ключі спільно використовуються, та як перевірка валідності введеного вручну ключа або відновленого з резервної копії. Зверніться до опису об'єктів для специфічних типів ключів за алгоритмами KCV.

Властивості `СКА_CHECK_VALUE`:

1. Для двох ключів, які мають криптографічно ідентичні значення, ці атрибути повинні бути ідентичними.
2. `СКА_CHECK_VALUE` не повинні використовуватися для отримання будь-якої частини ключа.
3. Не-унікальність. Два різні ключі можуть мати ті ж `СКА_CHECK_VALUE`.

Атрибут є не обов'язковим, але якщо ця функція підтримується, значення атрибута завжди повинно поставлятися бібліотекою, незалежно від того чи об'єкт створюється чи виробляється (derived). Атрибут повинен надаватися, навіть якщо операції шифрування для ключа заборонено (тобто, коли `СКА_ENCRYPT` встановлений в `СК_FALSE`).

Якщо значення поставляється в шаблон застосування (допускається, але ніколи не вимагається), то, якщо ця функція підтримується, то значення повинно відповідати тому, що бібліотека обчислює, чи бібліотека повинна повертати `СКR_ATTRIBUTE_VALUE_INVALID`. Якщо бібліотека не підтримує атрибут, то він повинен ігноруватися. Дозвіл цього атрибута в шаблоні, таким чином, не завдає шкоди і дозволяє обробляти будь-які інші атрибути для цілей «обгорнення» та «розгорнення» ключів.

Генерування KCV можна уникнути за рахунок застосування поставки атрибута в шаблоні, як запису, що не має значення (0 довжини). Застосування може запитати значення в будь-який час, як і будь-який інший атрибут, за допомогою функції `C_GetAttributeValue`. Функції `C_SetAttributeValue` можуть бути використані для знищення атрибутів, поставляючи запис, що не має значення (0 довжини).

Якщо інше не передбачено для визначення `СКА_CHECK_VALUE`, то значення цього атрибута походить від ключового об'єкту, узявши перші три байти шифрування одного блоку нульових (0x00) байт, використовуючи режим шифрування за замовчуванням (наприклад, ECB), пов'язаний з ключем типу секретний ключ.

3.8. Об'єкти «Механізм»

Цей розділ визначає об'єкт класу СКО_MECHANISM для типу СК_OBJECT_CLASS, як використовується в СКА_CLASS атрибутах об'єктів.

Об'єкти «Механізм» надають інформацію про механізми, що підтримується пристроєм, за допомогою СК_MECHANISM_INFO структури.

Під час пошуку об'єктів за допомогою C_FindObjectsInit та C_FindObjects, об'єкти «Механізм» не будуть повертатися, якщо СКА_CLASS атрибут в шаблоні має значення СКО_MECHANISM. Це захищає застосування, написані для попередніх версій Cryptoki від пошуку об'єктів, які вони не розуміють.

Загальні атрибути механізмів наведено у таблиці:

Таблиця 3.18. Загальні атрибути механізмів

Атрибут	Тип даних	Значення
СКА_MECHANISM_TYPE	СК_MECHANISM_TYPE	Тип об'єкту «Механізм»

Атрибут СКА_MECHANISM_TYPE може бути не встановлено.

4. Функції

4.1. Загальна інформація

Cryptoki- функції організовані за такими категоріями:

- функції загального призначення (4 функції)
- функції управління слотом і токеном (9 функцій)
- функції управління сесіями (8 функцій)
- функції управління об'єктами (9 функцій)
- функції зашифрування (4 функції)
- функції розшифрування (4 функції)
- функції хешування повідомлень (5 функцій)
- функції підписання та MAC-функції (6 функцій)
- функцій перевіряння підписів і MAC (6 функцій)
- криптографічні функції подвійного призначення (4 функції)
- функції керування ключами (5 функцій)
- функції генерування випадкових чисел (2 функції)
- функції управління паралельними функціями (2 функції)

На додаток до цих функцій, Cryptoki може застосовувати функції зворотного виклику для повідомлення додатків про застосування певних подій, і може також використовувати функції для обробки mutex-об'єктів для безпечного багато потокового доступу до бібліотек.

Виконання виклику функції Cryptoki є «в цілому все або нічого», тобто, виклик функції виконує або всю свою мету, або нічого взагалі:

- Якщо функція Cryptoki виконується успішно, то вона повертає значення `SKR_OK`.
- Якщо Cryptoki функція не виконується успішно, то вона повертає деяке значення, відмінне від `SKR_OK`, та токен знаходиться в тому ж стані, як це було до виклику функції. Якщо виклик функції повинен був змінити вміст певної адреси пам'яті на хост-комп'ютері, то ці адреси пам'яті можуть бути змінені, незважаючи на невиконання функції.
- Незвичайними обставинами є такі, що функція може не виконатися і повернути значення `SKR_GENERAL_ERROR`. Коли це відбувається, то токен та/або хост-комп'ютер може бути в нестійкому стані, і цілі функції можуть бути досягнуті частково.

Cryptoki-бібліотека може не підтримувати всі функції у Cryptoki API. Однак, навіть функції, що не підтримуються, повинні мати «заглушку» в бібліотеці, яка просто повертає значення `SKR_FUNCTION_NOT_SUPPORTED`. Запис цієї функції в структуру `CK_FUNCTION_LIST` бібліотеки (як отримана за допомогою `C_GetFunctionList`) повинен вказувати на цю «заглушку» функції.

4.2. Повернені функціями значення

Інтерфейс Cryptoki здатен обробляти велику кількість функцій і повернених значень. Тут надається визначення поведінки функцій Cryptoki достатньо повно; однак, можуть бути і деякі розходження. Наприклад, може бути, що певний код помилки, що повинен би відноситися до певної функції Cryptoki, фактично не зазначено в описі цієї функції, як можливий код помилки.

4.2.1. Універсальні повернені значення Cryptoki-функцій

Будь-яка функція Cryptoki може повертати будь-яке з наступних значень:

- **CKR_GENERAL_ERROR**: є критичною помилкою. У гіршому випадку, можливо, що функція виконана лише частково і комп'ютер та/або токен знаходиться в нестійкому/ невідповідному (inconsistent) стані.
- **CKR_HOST_MEMORY**: хост комп'ютер, на якому запущена Cryptoki бібліотека, не має достатньо пам'яті, щоб виконати необхідну функцію.
- **CKR_FUNCTION_FAILED**: необхідна функція не може бути виконана, але деталізується інформація про те, чому виникла ця помилка. Якщо під час сесії функція не виконується, можливо, що структура CK_SESSION_INFO, яку можна отримати шляхом виклику функції C_GetSessionInfo, буде містити корисну інформацію про те, що відбулося в полі ulDeviceError. У будь-якому разі, хоча виклик функції не був успішним, ситуація не є обов'язково безнадійною, як це може бути, коли повертається значення CKR_GENERAL_ERROR. Залежно від причини помилки, можливо, що буде ще спроба викликати цю ж саму функцію знову і вона завершиться успішно.
- **CKR_OK**: функцію виконано успішно. Технічно, CKR_OK не є досить «універсальним» повернутим значенням; зокрема, функції C_GetFunctionStatus та C_CancelFunction не можуть повертати значення CKR_OK.

Відносні пріоритети цих помилок наведено у порядку, що зазначений вище, наприклад, якщо будь-яке зі значень CKR_GENERAL_ERROR або CKR_HOST_MEMORY може відповідати певній помилці, то слід повернути значення CKR_GENERAL_ERROR.

4.2.2. Повернені значення Cryptoki-функції для функцій, що використовують дескриптори сесії

Будь-яка Cryptoki функція, яка має дескриптор (handle) сесії як один зі своїх аргументів (тобто, будь-яка Cryptoki функція за винятком C_Initialize, C_Finalize, C_GetInfo, C_GetFunctionList, C_GetSlotList, C_GetSlotInfo, C_GetTokenInfo, C_WaitForSlotEvent, C_GetMechanismList, C_GetMechanismInfo, C_InitToken, C_OpenSession та C_CloseAllSessions) може повернути такі значення:

- **CKR_SESSION_HANDLE_INVALID**: специфікований дескриптор сесії був неприпустимим *тоді, коли була викликана функція*. Зверніть увагу, що це

може статися, якщо токен сесії видаляється перед викликом функції, оскільки перед видаленням токен закриває всі його сесії.

- **CKR_DEVICE_REMOVED**: токен був вилучений з його слоту *під час виконання функції*.

- **CKR_SESSION_CLOSED**: сесію було завершено/ закрито *під час виконання функції*. Зверніть увагу, що поведінка Cryptoki є невизначеною, якщо декілька потоків застосування спробують одночасно отримати доступ до спільної сесії Cryptoki. Таким чином, немає насправді ніякої гарантії, що виклик функції може коли-небудь повернути значення **CKR_SESSION_CLOSED** — якщо один потік використовує сесію, коли інший потік закриває цю сесію, то є екземпляр декількох потоків, що використовують одночасно доступ до загальної сесії.

Відносні пріоритети цих помилок наведено у порядку, що зазначено вище, наприклад, якщо будь-яке із значень **CKR_SESSION_HANDLE_INVALID** або **CKR_DEVICE_REMOVED** може відповідати певній помилці, то слід повернути значення **CKR_SESSION_HANDLE_INVALID**.

На практиці, часто це не є вирішальним (або можливим) для бібліотеки Cryptoki, щоб робити відмінність між токеном, що видаляється перед викликом функції і токеном, що видаляється під час виконання функції.

4.2.3. Повернені значення Cryptoki-функції для функцій, що застосовують токен

- Будь-яка Cryptoki-функція, яка використовує певний токен (тобто, будь-яка Cryptoki-функція за винятком **C_Initialize**, **C_Finalize**, **C_GetInfo**, **C_GetFunctionList**, **C_GetSlotList**, **C_GetSlotInfo** або **C_WaitForSlotEvent**) може повернути будь-яке з наступних значень:

- **CKR_DEVICE_MEMORY**: токен не має достатньо пам'яті, щоб виконати необхідну функцію.

- **CKR_DEVICE_ERROR**: деякі проблеми виникають з токеном та/або слотом. Цей код помилки може бути повернутий не тільки для функцій, вказаних вище; але може й для **C_GetSlotInfo** бути повернуто значення **CKR_DEVICE_ERROR**.

- **CKR_TOKEN_NOT_PRESENT**: токен не був присутній у своєму слоті *під час виклику функції*.

- **CKR_DEVICE_REMOVED**: токен був вилучений зі свого слоту *під час виконання функції*.

Відносні пріоритети цих помилок наведено у порядку, що зазначено вище, наприклад, якщо будь-яке із значень **CKR_DEVICE_MEMORY** або **CKR_DEVICE_ERROR** може відповідати певній помилці, то слід повернути значення **CKR_DEVICE_MEMORY**.

На практиці, часто це не є вирішальним (або можливим) для бібліотеки Cryptoki, щоб робити відмінність між токеном, що видаляється перед викликом функції і токеном, що видаляється під час виконання функції.

4.2.4. Інші повернені значення Cryptoki-функції

Описи інших повернутих значень функції Cryptoki наведено нижче. За винятком випадків, вказаних в описі конкретних кодів помилок, в цілому не має ніяких особливих пріоритетів серед помилок, зазначених нижче. Тобто, якщо більше ніж один код помилки може застосовуватися при виконанні функції, то функція може повертати будь-який відповідний код помилки.

- **CKR_ARGUMENTS_BAD**: це загальний код помилки, який вказує, що аргументи, які застосовуються у функції Cryptoki, з якихось причин не підходять.

- **CKR_ATTRIBUTE_READ_ONLY**: була зроблена спроба встановити значення для атрибуту, яке не може бути встановлено або яке не може бути змінено застосуванням. Див далі щодо отримання додаткової інформації.

- **CKR_ATTRIBUTE_SENSITIVE**: була здійснена спроба отримати значення атрибута об'єкта, яка не може бути задоволена, оскільки об'єкт є або чутливим (*sensitive*) або таким, що не вилучається (*unextractable*).

- **CKR_ATTRIBUTE_TYPE_INVALID**: неправильний тип атрибута, вказаний у шаблоні. Див далі щодо отримання додаткової інформації.

- **CKR_ATTRIBUTE_VALUE_INVALID**: неприпустиме значення, вказане для певного атрибута у шаблоні. Див далі щодо отримання додаткової інформації.

- **CKR_BUFFER_TOO_SMALL**: вихідне значення функції занадто велике для розміщення у відповідному буфері.

- **CKR_CANT_LOCK**: це значення може бути повернуто лише функцією C_Initialize. Це означає, що тип фіксації/блокування (*locking*), необхідний програмі для безпеки потоків, не доступний в цій бібліотеці і, таким чином, програма не може використовувати цю бібліотеку у вказаній формі.

- **CKR_CRYPTOKI_ALREADY_INITIALIZED**: це значення може бути повернуто лише функцією C_Initialize. Це означає, що бібліотека Cryptoki вже ініціалізована (попереднім викликом функції C_Initialize, що не має відповідного виклику функції C_Finalize).

- **CKR_CRYPTOKI_NOT_INITIALIZED**: це значення може бути повернуто будь-якою функцією, окрім C_Initialize і C_GetFunctionList. Це означає, що функція не може бути виконана, оскільки бібліотека Cryptoki ще не була ініціалізована за допомогою виклику функції C_Initialize.

- **CKR_DATA_INVALID**: введені дані криптографічної операції є недійсними. Повернене значення має більш низький пріоритет, ніж **CKR_DATA_LEN_RANGE**.

- **CKR_DATA_LEN_RANGE**: введені дані криптографічної операції мають неправильну довжину. Залежно від механізму операції, це може означати, що незашифрований текст дуже короткий, занадто довгий або не є кратним деякому розміру блоку. Повернене значення має більш високий пріоритет, ніж **CKR_DATA_INVALID**.

- **CKR_ENCRYPTED_DATA_INVALID**: зашифровані данні, що вводяться для операції розшифрування, визначені як помилковий зашифрований текст. Повернене значення має більш низький пріоритет, ніж **CKR_ENCRYPTED_DATA_LEN_RANGE**.

- **CKR_ENCRYPTED_DATA_LEN_RANGE**: зашифровані данні, що вводяться для операції розшифрування, визначені як помилковий зашифрований текст виключно на основі його довжини. Залежно від механізму операції, це може означати, що зашифрований текст занадто короткий, дуже великий або не кратний деякому розміру блоку. Повернене значення має більш високий пріоритет, ніж **CKR_ENCRYPTED_DATA_INVALID**.

- **CKR_FUNCTION_CANCELED**: функція була скасована під час її виконання. Це відбувається для криптографічної функції, якщо функція робить зворотній виклик програми **CKN_SURRENDER** та повертає **CKR_CANCEL** (див. **CKR_CANCEL**). Це відбувається і для функції, яка виконує вхід за PIN-кодом через захищений шлях. Метод, що використовується для скасування операції входу за PIN-кодом через захищений шлях є залежним від пристрою.

- **CKR_FUNCTION_REJECTED**: запит підпису відхилено користувачем.

- **CKR_INFORMATION_SENSITIVE**: необхідна інформація не може бути отримана, тому що токен вважає її чутливою і не в змозі або не повинен розкрити її.

- **CKR_KEY_FUNCTION_NOT_PERMITTED**: була зроблена спроба використати ключ для криптографічних цілей, але атрибути ключа не задані, щоб це зробити. Наприклад, якщо використовується ключ для виконання шифрування, такий ключ повинен мати свій атрибут **CKA_ENCRYPT** встановлений у **CK_TRUE** (той факт, що ключ повинен мати атрибут **CKA_ENCRYPT** визначає, що ключ не може бути особистим ключем). Це повернене значення має більш низький пріоритет, ніж **CKR_KEY_TYPE_INCONSISTENT**.

- **CKR_KEY_HANDLE_INVALID**: зазначений дескриптор ключа є неприпустимим. Це може бути у випадку, якщо зазначений дескриптор є припустимим дескриптором для об'єкта, який не є ключем. Слід підкреслити, що 0 ніколи не є дійсним дескриптором ключа.

- **CKR_KEY_NOT_WRAPPABLE**: незважаючи на те, що заданий особистий або секретний ключ не має атрибуту **CKA_UNEXTRACTABLE**, встановленого у **CK_TRUE**, **Cryptoki** (або токен) не може «обгорнути»/зашифрувати ключ як це потрібно (можливо токен може тільки зашифрувати ключ конкретного типу, але ключ «обгорнення» не відноситься до цих типів). Порівняйте з **CKR_KEY_UNEXTRACTABLE**.

- **CKR_KEY_SIZE_RANGE**: хоча потрібна криптографічна операція в принципі може бути проведена, ця бібліотека **Cryptoki** (або токен) не може насправді зробити це, тому що розмір наданого ключа знаходиться поза межами діапазону розмірів ключів, що може оброблятися.

- **CKR_KEY_TYPE_INCONSISTENT**: зазначений ключ не відноситься до правильного типу ключів для використання зазначеним механізмом. Це повернене значення має більш високий пріоритет, ніж **CKR_KEY_FUNCTION_NOT_PERMITTED**.

- **CKR_KEY_UNEXTRACTABLE**: зазначений особистий або секретний ключ не може бути «обгорнуто»/зашифровано тому, що його атрибут **CKA_UNEXTRACTABLE** має значення **CK_TRUE**. Порівняйте з **CKR_KEY_NOT_WRAPABLE**.

- **CKR_MECHANISM_INVALID**: невірний механізм був вказаний для криптографічної операції. Цей код помилки є відповідним поверненим значенням, якщо був обраний невідомий механізм або зазначений механізм не може бути використано в обраному токени для обраної функції.

- **CKR_MECHANISM_PARAM_INVALID**: невірні параметри були застосовані у механізмі, обраному для криптографічної операції. Які значення параметрів підтримує заданий механізм може варіюватися в залежності від токени.

- **CKR_NEED_TO_CREATE_THREADS**: це значення може бути повернуто тільки функцією **C_Initialize**. Воно повертається, коли є дві умови:

1. Програма викликає **C_Initialize** шляхом, який показує бібліотеку **Cryptoki**, що виконання викликів програмних потоків (threads) для бібліотеки не може використати власні (native) методи операційної системи, щоб створити нові потоки.
2. Бібліотека не може функціонувати належним чином, не будучи в змозі створити нові потоки вищезгаданим способом.

- **CKR_OBJECT_HANDLE_INVALID**: зазначений дескриптор об'єкту є неприпустимим. Слід підкреслити, що 0 ніколи не є дійсним дескриптором об'єкта.

- **CKR_OPERATION_ACTIVE**: існує активна операція (або поєднання активних операцій), яка запобігає/не дозволяє **Cryptoki** активувати зазначену операцію. Наприклад, активна операція пошуку об'єкту буде перешкоджати **Cryptoki** активувати операцію шифрування з **C_EncryptInit**. Або активна операція хешування і активна операція шифрування буде перешкоджати **Cryptoki** активувати операцію підписування. Або токен, який не підтримує одночасне виконання двох криптографічних операцій у сесії (див. опис прапора **CKF_DUAL_CRYPT_OPERATIONS** в структурі **CK_TOKEN_INFO**), активна операція підписування буде перешкоджати **Cryptoki** активувати операцію шифрування.

- **CKR_OPERATION_NOT_INITIALIZED**: Не існує активної операції відповідного типу у зазначеній сесії. Наприклад, програма не може викликати **C_Encrypt** у сесії, не викликавши до цього **C_EncryptInit**, перш ніж активувати операцію шифрування.

- **CKR_PIN_EXPIRED**: вказаний PIN втратив чинність і необхідна операція не може бути здійснена, доки **C_SetPIN** не буде викликана для зміни значення PIN-коду. PIN Звичайного користувача втратив чинність або залежить від типу токени.

- **CKR_PIN_INCORRECT**: вказаний PIN-код є невірним, тобто не відповідає PIN-коду, який зберігається у токени. У більш загальному випадку - коли аутентифікація у токени включає в себе щось інше, ніж використання PIN - спроба аутентифікації користувача не буде успішною.

- **CKR_PIN_INVALID**: вказаний PIN-код містить неприпустимі символи. Цей повернений код використовується тільки для функцій, які намагаються застосувати PIN-код.

- **CKR_PIN_LEN_RANGE**: вказаний PIN-код є занадто довгим чи занадто коротким. Цей повернений код використовується тільки для функцій, які намагаються застосувати PIN-код.

- **CKR_PIN_LOCKED**: вказаний PIN-код є "заблокованим" і не може бути використаний. Тобто, тому що була досягнута деяка конкретна кількість невдалих спроб аутентифікації, токен не дає дозволу на подальші спроби аутентифікації. У залежності від токена, вказаний PIN може або не може залишатися заблокованим на невизначений термін.

- **CKR_SESSION_COUNT**: це значення може бути повернуто лише функцією `C_OpenSession`. Воно означає, що спроба відкрити сесію була невдалою, або те, що токен вже відкрив багато сесій, або те, що токен вже має занадто багато відкритих сесій читання/ запису.

- **CKR_SESSION_EXISTS**: це значення може бути повернуто лише `C_InitToken`. Воно означає, що сесія з токеном вже відкрита і тому токен не може бути ініціалізований.

- **CKR_SESSION_PARALLEL_NOT_SUPPORTED**: зазначений токен не підтримує паралельні сесії. Це спадковий код помилки - у Cryptoki версії 2.01 і вище, токен *не* підтримує паралельні сесії. Може бути повернуто лише функцією `C_OpenSession` і воно повертається тільки при викликанні `C_OpenSession` особливим [застарілим *-deprecated*] шляхом.

- **CKR_SESSION_READ_ONLY**: зазначеній сесії не вдалося досягти бажаної дії, тому що це тільки сесія для читання. Це повернене значення має більш низький пріоритет, ніж **CKR_TOKEN_WRITE_PROTECTED**.

- **CKR_SESSION_READ_ONLY_EXISTS**: сесія тільки для читання вже існує і тому SO не може виконати аутентифікацію.

- **CKR_SESSION_READ_WRITE_SO_EXISTS**: Сесія SO читання/запису (*read/write SO session*) вже існує і тому тільки для сесії читання не може бути відкрита.

- **CKR_SIGNATURE_LEN_RANGE**: підпис/ MAC є недійсним виключно на основі його довжини. Це повернене значення має більш вищий пріоритет, ніж **CKR_SIGNATURE_INVALID**.

- **CKR_SIGNATURE_INVALID**: підпис / MAC є недійсним. Це повернене значення має нижчий пріоритет, ніж **CKR_SIGNATURE_LEN_RANGE**.

- **CKR_SLOT_ID_INVALID**: зазначений ідентифікатор слоту є недійсним.

- **CKR_TEMPLATE_INCOMPLETE**: шаблон, що використовується для створення об'єкту, є неповним і у ньому не вистачає необхідних атрибутів. Див. далі щодо отримання додаткової інформації.

- **CKR_TEMPLATE_INCONSISTENT**: шаблон, що використовується для створення об'єкту має суперечливі атрибути. Див. далі щодо отримання додаткової інформації.

- **CKR_TOKEN_NOT_RECOGNIZED**: бібліотека Cryptoki та/або слот не може розпізнати токен у слоті.

- **CKR_TOKEN_WRITE_PROTECTED**: необхідна дія не може бути виконана, оскільки токен є захищеним від запису. Це повернене значення має більш високий пріоритет, ніж **CKR_SESSION_READ_ONLY**.

- **CKR_UNWRAPPING_KEY_HANDLE_INVALID**: це значення може бути повернуто лише функцією **C_UnwrapKey**. Воно означає, що дескриптор ключа, що використовується для «розгорнення»/ розшифрування іншого ключа, є недійсним.

- **CKR_UNWRAPPING_KEY_SIZE_RANGE**: це значення може бути повернуто лише функцією **C_UnwrapKey**. Воно вказує, що необхідна операція «розгорнення»/ розшифрування в принципі могла б бути виконана; ця бібліотека **Cryptoki** (або токен) не може насправді виконати це, тому що розмір ключа, що використовується, знаходиться поза межами діапазону розмірів ключів, який може оброблятися.

- **CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT**: це значення може бути повернуто лише функцією **C_UnwrapKey**. Воно вказує, що тип ключа, що використовується для «розгорнення»/ розшифрування іншого ключа, не відповідає механізму, передбаченому для «розгорнення»/ розшифрування.

- **CKR_USER_ALREADY_LOGGED_IN**: це значення може бути повернуто лише функцією **C_Login**. Вона вказує, що зазначений користувач не може увійти до сесії, тому що він вже увійшов до сесії раніше. Наприклад, якщо програма має відкриту **SO** сесію, і є спроба увійти в сесію, то буде отримано цей код помилки.

- **CKR_USER_ANOTHER_ALREADY_LOGGED_IN**: це значення може бути повернуто лише функцією **C_Login**. Вона вказує, що зазначений користувач не може увійти до сесії, тому що інший користувач вже увійшов до сесії. Наприклад, якщо програма має відкриту **SO** сесію, і є спроба Звичайного користувача увійти до цієї сесії, то він отримає цей код помилки.

- **CKR_USER_NOT_LOGGED_IN**: бажана дія не може бути виконана, оскільки відповідний користувач (або конкретний користувач) не пройшли аутентифікацію. Одним із прикладів є те, що неможливо вийти з сесії, якщо не було здійснено вхід до цієї сесії. Іншим прикладом є те, що особистий об'єкт не може бути створений токеном, доки сесія намагається створити його як Звичайний користувач. Останній приклад є те, що криптографічні операції для конкретних токенів не можуть бути виконані, доки Звичайний користувач є авторизованим.

- **CKR_USER_PIN_NOT_INITIALIZED**: це значення може бути повернуто лише функцією **C_Login**. Воно вказує, що **PIN**-код Звичайного користувача ще не був ініціалізований за допомогою функції **C_InitPIN**.

- **CKR_USER_TOO_MANY_TYPES**: були здійснені спроби різними користувачами одночасно виконати ініціалізацію у токені більше разів, ніж токен та/або бібліотека дозволяє. Наприклад, якщо яка-небудь програма має відкриту **SO** сесію, а інша програма здійснює спробу ініціалізувати звичайного користувача у сесії, то спроба може повернути цю помилку. Однак, це не є обов'язковим. Тільки якщо різні користувачі не можуть бути підтримані, то функція **C_Login** повинна повернути це значення. Зауважимо, що цей код помилки генерується для справжніх багатокористувацьких токенів.

- `CKR_USER_TYPE_INVALID`: неприпустиме значення, що визначено як `CK_USER_TYPE`. Чинними типами є `CKU_SO`, `CKU_USER`, і `CKU_CONTEXT_SPECIFIC`.

- `CKR_WRAPPED_KEY_INVALID`: це значення може бути повернуто лише функцією `C_UnwrapKey`. Воно вказує, що наданий «загорнутий»/зашифрований ключ є недійсним. Якщо викликається `C_UnwrapKey` для «розгорнення»/розшифрування конкретного типу ключа (наприклад, певний визначений тип ключа вказується в шаблоні для `C_UnwrapKey`) і ключ, наданий `C_UnwrapKey`, не розпізнається як ключ відповідного типу, то `C_UnwrapKey` повинна повернути `CKR_WRAPPED_KEY_INVALID`. Це повернене значення має нижчий пріоритет, ніж `CKR_WRAPPED_KEY_LEN_RANGE`.

- `CKR_WRAPPED_KEY_LEN_RANGE`: це значення може бути повернуто тільки `C_UnwrapKey`. Воно вказує, що наданий «обгорнений»/ зашифрований ключ може бути визнано недійсним виключно на основі його довжини. Це повернене значення має більш високий пріоритет, ніж `CKR_WRAPPED_KEY_INVALID`.

- `CKR_WRAPPING_KEY_HANDLE_INVALID`: це значення може бути повернуто тільки `C_WrapKey`. Воно означає, що дескриптор ключа, визначений для використання при зашифруванні іншого ключа, є недійсним.

- `CKR_WRAPPING_KEY_SIZE_RANGE`: це значення може бути повернуто лише `C_WrapKey`. Воно вказує, що, необхідна операція «обгорнення» в принципі могла б бути здійснена; бібліотека `Cryptoki` (або токен) не може насправді зробити це, тому що розмір наданого ключа «обгорнення» знаходиться поза межами діапазону розмірів ключів, що можуть оброблятися.

- `CKR_WRAPPING_KEY_TYPE_INCONSISTENT`: це значення може бути повернуто лише `C_WrapKey`. Воно вказує, що тип ключа, визначеного для «обгорнення» іншого ключа, не відповідає механізму, передбаченому для «обгорнення».

4.2.5. Про відносні пріоритети помилок `Cryptoki`

Загалом, коди помилок з розділу 2.10.1 (окрім `CKR_OK`) мають більш високий пріоритет над кодами помилок з розділу 2.10.2, які, в свою чергу, мають пріоритет над кодами помилок з 2.10.3 і так далі.

Наслідком цього є те, що функції, які використовують дескриптор сесії (тобто, більшість функцій) ніколи не повертають код помилки `CKR_TOKEN_NOT_PRESENT` – замість цього вони повертають `CKR_SESSION_HANDLE_INVALID`. Крім цих пріоритетів, якщо застосовується до результату виклику `Cryptoki` більше одного коду помилок, то можуть бути повернені будь-які із кодів помилок, що мають місце. Винятки з цього правила будуть явно зазначені в описах функцій.

4.2.6. Конвенція щодо значень, які повертають функції до буфера змінної довжини

Ряд функцій, визначених у Cryptoki, повертають значення, вироблені певними криптографічними механізмами. Кількість вихідних даних, що повертається цими функціями, повертається як буфер змінної довжини. Прикладом функції такого роду є `C_Encrypt`, яка вимагає повного буфера для зашифрованого тексту.

Ці функції мають певну спільну угоду (Конвенцію) про виклик, який тут описується: Двома із аргументів функції є покажчик на вихідний буфер (скажімо, `pBuf`) і покажчик на довжину вироблених вихідних даних (скажімо `pulBufLen`). Для програми є два шляхи викликати такі функції:

1. Якщо `pBuf` є `NULL_PTR`, то все, що функція поверне (в `*pulBufLen`) – це число байт, якого було б достатньо, щоб утримувати криптографічні вихідні дані, вироблені функцією від вхідних даних. Це число може дещо перевищувати точне число необхідних байтів, але не повинно перевищувати його на значну величину. Функцією повертається `CKR_OK`.

2. Якщо `pBuf` є non-`NULL_PTR`, то `*pulBufLen` повинен містити розмір в байтах буфера, на який вказує `pBuf`. Якщо цей буфер досить великий для зберігання криптографічно вироблених вихідних даних, то ці вихідні дані знаходяться там, і функцією повертається `CKR_OK`. Якщо буфер недостатньо великий, то повертається `CKR_BUFFER_TOO_SMALL`. У будь-якому випадку, `*pulBufLen` містить точне число байт, необхідних для криптографічних вихідних даних, вироблених функцією від вхідних даних.

Всі функції, які використовують вищезазначену конвенцію буде явного згадування про це нам.

Криптографічні функції, які повертають вихідні в змінної довжини буфера повинен завжди повертати стільки вихід, як можна обчислити з того, що було, переданих в них до цих пір. Як приклад, розглянемо сесії, яка виконує декілька частин розшифровки роботи з DES шифром блок ланцюжка режим заповнення PKCS. Припустимо, що спочатку 8 байт шіфротекста передаються функції `C_DecryptUpdate`. Розмір блоку DES становить 8 байт, але оббивка PKCS робить неясним на даному етапі чи зашифрований текст був підготовлений з шифруванням 0-байт рядка, або зашифрувати деяку рядок довжиною не менше 8 байт. Тому виклик `C_DecryptUpdate` повинна повертати 0 байт тексту. Якщо одного додаткового байта зашифрованого поставляється наступний виклик `C_DecryptUpdate`, те, що виклик повинен повертати 8 байт тексту (один повний блок DES).

4.3. Функції загального призначення

Cryptoki повинна надавати такі функції загального призначення:

4.3.1. C_Initialize

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

Функція C_Initialize ініціалізує бібліотеку Cryptoki.

Аргумент *pInitArgs* або має значення NULL_PTR, або вказує на CK_C_INITIALIZE_ARGS структуру, яка містить інформацію про те, яким чином бібліотека повинна мати справу з багато-потокowym доступом. Якщо програма не отримуватиме доступ до Cryptoki через кілька зв'язаних одночасно каналів, вона взагалі може постачати значення NULL_PTR для C_Initialize (наслідки постачання цього значення буде пояснено нижче).

Якщо *pInitArgs* не є NULL_PTR, то C_Initialize повинна призначити його CK_C_INITIALIZE_ARGS_PTR і потім із покажчика отримати CK_C_INITIALIZE_ARGS полів *CreateMutex*, *DestroyMutex*, *LockMutex*, *UnlockMutex*, *flags* та *pReserved*.

Для цієї версії Cryptoki значення *pReserved* повинно бути NULL_PTR; якщо це не так, то C_Initialize повинна повернути значення CKR_ARGUMENTS_BAD.

Якщо прапор CKF_LIBRARY_CANT_CREATE_OS_THREADS у полі прапорів встановлений, то це вказує, що програмі застосування, яка виконує виклик бібліотеки Cryptoki, не дозволяється використовувати вбудовані (native) виклики операційної системи для створення нових потоків (threads). Іншими словами, код бібліотеки не може створити власний потік. Якщо бібліотека не в змозі нормально функціонувати при цих обмеженнях, то C_Initialize повинні повернути значення CKR_NEED_TO_CREATE_THREADS.

Виклик C_Initialize визначає один з чотирьох різних способів підтримки багато-потокowego доступу через значення прапора CKF_OS_LOCKING_OK у полі прапорів і значень *CreateMutex*, *DestroyMutex*, *LockMutex* і *UnlockMutex* полів покажчика функції:

1. Якщо не встановлено прапор і поля покажчика функції не застосовуються (тобто, всі вони мають значення NULL_PTR), це означає, що програма не буде доступною Cryptoki бібліотеці з кількома одночасно потоками.

2. Якщо прапор встановлено, і поля покажчика функції не застосовуються (тобто, всі вони мають значення NULL_PTR), то це означає, що програма буде виконувати багато-потокowy Cryptoki-доступ, та бібліотеці необхідно застосовувати вбудовані (native) примітиви операційної системи для забезпечення безпечного багато-потокowego доступу. Якщо бібліотека не в змозі це зробити, то C_Initialize повинна повернути значення CKR_CANT_LOCK.

3. Якщо не встановлено прапор, і поля покажчика функції застосовуються (тобто, всі вони мають значення non-NULL_PTR), то це означає, що програма буде виконувати багато-потокowy Cryptoki-доступ та бібліотека повинна використовувати надані покажчики функції mutex-обробки для забезпечення

безпечного багато-потокowego доступу. Якщо бібліотека не в змозі робити це, то `C_Initialize` повинна повернути значення `CKR_CANT_LOCK`.

4. Якщо прапор *встановлено*, і поля покажчика функції *застосовуються* (тобто, всі вони мають значення `non-NULL_PTR`), то це означає, що програма *буде* виконувати багато-потоківий Cryptoki-доступ та бібліотека повинна використовувати або вбудовані (*native*) примітиви операційної системи, або надані покажчики функції mutex-обробки для забезпечення безпечного багато-потокowego доступу. Якщо бібліотека не в змозі зробити це, *C_Initialize* повинна повернути значення `CKR_CANT_LOCK`.

Якщо деякі, але не всі з наданих покажчиків функції для `C_Initialize` мають значення `non-NULL_PTR`, то `C_Initialize` повинна повернути значення `CKR_ARGUMENTS_BAD`.

Виклик `C_Initialize` з *pInitArgs*, встановленим у `NULL_PTR`, розглядається як виклик `C_Initialize` з *pInitArgs*, яке вказує на `CK_C_INITIALIZE_ARGS`, що має *CreateMutex*, *DestroyMutex*, *LockMutex*, *UnlockMutex* та *pReserved* поля, які встановлено у `NULL_PTR`, і має поле прапорів із значенням 0.

`C_Initialize` повинно бути першим викликом Cryptoki, зробленим програмою, за винятком викликів `C_GetFunctionList`. Ця функція фактично робить реалізацію залежною; зазвичай, це може бути причиною, щоб Cryptoki ініціалізувала власні буфери внутрішньої пам'яті, або будь-який інші ресурси, що потрібні.

Якщо декілька програм використовують Cryptoki, то кожній з них необхідно викликати `C_Initialize`. Кожен виклик `C_Initialize` повинен бути успадкованим від одного виклику `C_Finalize`.

Повернені значення: `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`.

Приклад: див `C_GetInfo`.

4.3.2. `C_Finalize`

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

`C_Finalize` викликається для індикації того, що програма завершила роботу з бібліотекою Cryptoki. Цей виклик повинен бути останнім викликом Cryptoki, який зроблено програмою.

Параметр *pReserved* зарезервовано для майбутніх версій; для цієї версії він повинен бути встановлений у `NULL_PTR` (якщо `C_Finalize` викликається зі

значенням non-NULL_PTR для *pReserved*, то слід повернути значення CKR_ARGUMENTS_BAD.

Якщо кілька програм використовують Cryptoki, кожна з них повинна викликати C_Finalize. Кожному виклику C_Finalize з додатку повинен передувати один виклик C_Initialize; між двома викликами, програма може викликати інші функції Cryptoki.

Незважаючи на те, що параметри для C_Initialize в цілому можна застосовувати для безпечного багато-потокowego доступу до бібліотеки Cryptoki, режим C_Finalize є тим не менше невизначеним, якщо він викликається програмою, доки інші потоки програми здійснюють виклики Cryptoki. За винятком, коли потік викликає C_Finalize, поки інші потоки програми заблоковані функцією Cryptoki C_WaitForSlotEvent. Коли це відбудеться, заблоковані потоки розблоковуються і повертається значення CKR_CRYPTOKI_NOT_INITIALIZED. Див. C_WaitForSlotEvent.

Повернені значення:	CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.
---------------------	--

Приклад: див C_GetInfo.

4.3.3. C_GetInfo

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

C_GetInfo повертає загальну інформацію про Cryptoki.
Аргумент *pInfo* вказує розташування, де одержується інформація.

Повернені значення:	CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.
---------------------	--

Приклад:

```
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.version.major == 2) {
    /* Do lots of interesting cryptographic things with the token */
    .
    .
}

rv = C_Finalize(NULL_PTR);
assert(rv == CKR_OK);

```

4.3.4. C_GetFunctionList

```

CK_DEFINE_FUNCTION(CK_RV, C_GetFunctionList)(
    CK_FUNCTION_LIST_PTR_PTR ppFunctionList
);

```

`C_GetFunctionList` отримує покажчик на список бібліотеки Cryptoki для покажчиків функцій.

`ppFunctionList` вказує на значення, яке буде отримувати покажчик на структуру `CK_FUNCTION_LIST` бібліотеки, яка у свою чергу, містить покажчики функцій для всіх Cryptoki API процедур в бібліотеці. Покажчик, отриманий таким чином, може вказувати на область пам'яті, яка належить до бібліотеки Cryptoki і яка може або не може використовуватися для запису. Так це, чи ні, не повинно бути ніякої спроби, щоб записати у цю область пам'яті.

`C_GetFunctionList` є єдиною Cryptoki-функцією, яку програма може викликати перед викликом функції `C_Initialize`. Вона надається для того, щоб зробити процедуру простішою та швидшою для програм, що використовують спільні бібліотеки Cryptoki, та застосовують більше однієї Cryptoki бібліотеки одночасно.

Повернені значення: `CKR_ARGUMENTS_BAD`, `CKR_FUNCTION_FAILED`, `CKR_GENERAL_ERROR`, `CKR_HOST_MEMORY`, `CKR_OK`.

Приклад:

```

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);
```

4.4. Функції управління слотом та токеном

Cryptoki надає такі функції для управління слотом та токеном:

4.4.1. C_GetSlotList

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

C_GetSlotList використовується для отримання списку слотів у системі.
tokenPresent вказує на те, чи отриманий список включає тільки слоти з існуючими токенами (CK_TRUE), або всі слоти (CK_FALSE);
pulCount вказує розташування, де отримується номер слотів.

Є два шляхи, якими застосування викликають C_GetSlotList:

1. Якщо *pSlotList* має значення NULL_PTR, то C_GetSlotList тільки повертає (в * *pulCount*) число слотів, без повернення списку слотів. Вміст буферу C_GetSlotList не має значення в даному випадку, і виклик повертає значення CKR_OK.

2. Якщо *pSlotList* не NULL_PTR, то * *pulCount* повинен містити розмір (з точки зору CK_SLOT_ID елементів) буфера, що має покажчик *pSlotList*. Якщо буфер достатньо великий, щоб містити список слотів, то список повертається у ньому, а також повертається CKR_OK. Якщо ні, то виклик C_GetSlotList повертає значення CKR_BUFFER_TOO_SMALL. У будь-якому випадку, значення **pulCount* встановлюється для утримання числа слотів.

Оскільки C_GetSlotList не виділяє будь-якого власного простору, то застосування часто буде викликати C_GetSlotList два рази (або іноді навіть декілька разів — якщо програма намагається отримати список усіх слотів існуючих токенів, то номер кожного слоту може, на жаль, змінюватися між тим, коли застосування запитує скільки таких слотів є, та коли запитує слоти безпосередньо). Однак, декілька викликів C_GetSlotList є далеко не обов'язковим.

Усі слоти, які C_GetSlotList повідомляє, повинні бути перевірені на валідність C_GetSlotInfo. Крім того, набір слотів, які доступні через Cryptoki-бібліотеку, перевіряються під час виклику C_GetSlotList для передбачення довжини списку (аргумент NULL pSlotList). Якщо застосування викликає C_GetSlotList із значенням non –NULL для pSlotList, а потім користувач додає або видаляє апаратні засоби (пристрої), то буде видимим та ефективним тільки змінений список слотів, якщо C_GetSlotList визивається знову зі значенням NULL. Навіть якщо C_GetSlotList успішно визивається таким шляхом, може або не може бути так, що змінений список слотів буде успішно визнаний в залежності від бібліотеки виконання. На деяких платформах, або для ранніх версій сумісних бібліотек PKCS11, він можливо буде необхідним для успішного виклику C_Initialize або для перезапуску всієї системи.

Повернені значення: CKR_ARGUMENTS_BAD,
CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED,
CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY,
CKR_OK.

Приклад:

```
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);

```

4.4.2. C_GetSlotInfo

```

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

```

C_GetSlotInfo отримує інформацію про той чи інший слот у системі.
slotID є ідентифікатором слоту;
pInfo вказує на місце (показчик), яке утримує інформацію слоту.

Повертаються значення: CKR_ARGUMENTS_BAD,
CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR,
CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY,
CKR_OK, CKR_SLOT_ID_INVALID.

Приклад: див. C_GetTokenInfo.

4.4.3. C_GetTokenInfo

```

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

```

C_GetTokenInfo отримує інформацію про окремий токен у системі.
slotID є ідентифікатором слоту;
pInfo вказує на місце (показчик), яке утримує інформацію слоту.

Повертаються значення: 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.

Приклад:

```
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);
}
```

4.4.4. C_WaitForSlotEvent

<pre>CK_DEFINE_FUNCTION(CK_RV, C_WaitForSlotEvent)(CK_FLAGS flags, CK_SLOT_ID_PTR pSlot, CK_VOID_PTR pReserved);</pre>
--

C_WaitForSlotEvent чекає на подію у слоті, наприклад, встановлення або видалення токена.

flags визначає був чи ні виклик C_WaitForSlotEvent блокований (тобто, чекає на виникнення події у слоті);
pSlot вказує на місце, де буде утримуватися ID слоту;
pReserved відведені для майбутніх версій; для цієї версії Cryptoki, повинно бути NULL_PTR.

Якщо присутній, то прапор, що застосовується у аргументі flags, буде CKF_DONT_BLOCK:

Кожна програма Cryptoki має прапор для кожного слоту, який використовується для відстеження чи не виникли будь-які нерозпізнані події у цьому слоті. Коли програма спочатку викликає C_Initialize, то кожен прапор події у слоті знімається. Кожного разу, коли відбувається подія у слоті, прапор узгоджується із слотом, в якому сталася подія.

Якщо викликається C_WaitForSlotEvent із встановленим прапором CKF_DONT_BLOCK в аргументі flags, і деякі прапори подій у слоті встановлено, то флаг події знімається і виклик повертає ID цього слоту у місце, що визначено у pSlot. Якщо більш ніж один прапор події у слоті встановлено під час виклику, то такий слот вибирається бібліотекою і його прапор події знімається та повертається ID слоту.

Якщо викликається C_WaitForSlotEvent із встановленим прапором CKF_DONT_BLOCK в аргументі flags, і не встановлено жодного прапору події у слоті, то виклик повертає значення CKR_NO_EVENT. У цьому випадку зміст розташування, що вказано у pSlot, невизначений.

Якщо викликається C_WaitForSlotEvent із встановленим прапором CKF_DONT_BLOCK в аргументі flags, то виклик відбувається як зазначено вище, за винятком того, що він буде блокований. Тобто, якщо немає прапору події у слоті під час виклику, то C_WaitForSlotEvent буде чекати, поки деякі прапори події не будуть встановлені. Якщо потік програми має виклик C_WaitForSlotEvent блокування, коли інший потік викликає C_Finalize, то виклик C_WaitForSlotEvent повертає значення CKR_CRYPTOKI_NOT_INITIALIZED.

Хоча параметри для C_Initialize в цілому можна застосовувати для безпечного багато-поточного доступу до бібліотеки Cryptoki, C_WaitForSlotEvent є винятком, у тому плані, що поведінка Cryptoki є невизначеною, якщо декілька потоків однієї програми одночасно викликають C_WaitForSlotEvent.

Повернені значення:	CKR_ARGUMENTS_BAD,
CKR_CRYPTOKI_NOT_INITIALIZED,	CKR_FUNCTION_FAILED,
CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_NO_EVENT, CKR_OK.	

Приклад:

```
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);
.

```

4.4.5. C_GetMechanismList

```

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

```

C_GetMechanismList використовується для отримання списку типів механізмів, що підтримуються токеном.

SlotID є ідентифікатором слота токена;

pulCount вказує розташування, де утримується число механізмів.

Є два шляхи, якими застосування викликають C_GetMechanismList:

1. Якщо pMechanismList має значення NULL_PTR, то C_GetMechanismList тільки повертає (в *pulCount) число механізмів, без повернення списку механізмів. Вміст *pulCount на вході для C_GetMechanismList не має значення в даному випадку і виклик повертає значення CKR_OK.

2. Якщо pMechanismList не NULL_PTR, то *pulCount повинен містити розмір буфера (з точки зору CK_MECHANISM_TYPE елементів), що вказує на pMechanismList. Якщо буфер достатньо великий, щоб умістити список механізмів, то список повертається у ньому, а також CKR_OK повертається. Якщо ні, то виклик C_GetMechanismList повертає значення CKR_BUFFER_TOO_SMALL. У будь-якому випадку, значення *pulCount встановлюється для утримання числа механізмів.

Оскільки C_GetMechanismList не виділяє будь-якого власного простору, то застосування часто буде змушене викликати C_GetMechanismList два рази. Однак, така поведінка є далеко не обов'язковою.

Повернені значення:	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.

Приклад:

```
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);  
}
```

4.4.6. C_GetMechanismInfo

<pre>CK_DEFINE_FUNCTION(CK_RV, C_GetMechanismInfo)(CK_SLOT_ID slotID, CK_MECHANISM_TYPE type, CK_MECHANISM_INFO_PTR pInfo);</pre>

C_GetMechanismInfo отримує інформацію про конкретний механізм, що підтримується токеном.

slotID є ідентифікатором слота токenu;

type є типом механізму;

pInfo вказує розташування, де утримується інформація про механізми.

Повернені значення: 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.

Приклад:

```
CK_SLOT_ID slotID;  
CK_MECHANISM_INFO info;  
CK_RV rv;  
  
. .  
/* Get information about the CKM_MD2 mechanism for this token */  
rv = C_GetMechanismInfo(slotID, CKM_MD2, &info);  
if (rv == CKR_OK) {  
    if (info.flags & CKF_DIGEST) {  
        . .  
    }  
}
```

4.4.7. C_InitToken

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

C_InitToken ініціалізує токен.

slotID є ідентифікатором слота токена;

pPin вказує на початковий SO PIN, (який не повинен бути нульовим);

ulPinLen є довжиною PIN-коду у байтах;

pLabel вказує на 32-байтову мітку токена, (яка повинна бути доповнена пустими символами (пробілами) і яка не може мати нульове значення). Цей стандарт дозволяє значенню PIN-кода містити будь-які припустимі символи UTF8, але токен може накласти підмножину обмежень.

Якщо токен не ініціалізовано (тобто він є новим з заводу), то параметр pPin приймає початкове значення SO PIN. Якщо токен заново ініціалізовано, то параметр pPin, перевіряється на існуючих SO PIN для дозволу операції ініціалізації. В обох випадках SO PIN є значенням pPin після успішного завершення функції. Якщо SO PIN втрачається, то токен необхідно буде заново ініціалізувати за допомогою механізму, що є поза межами цієї специфікації.

Прапор CKF_TOKEN_INITIALIZED у CK_TOKEN_INFO структурі вказує дії, які будуть результатом виклику C_InitToken. Якщо цей прапор встановлено,

то токен буде заново ініціалізовано і клієнтові потрібно вказати існуючий пароль SO у pPin.

Коли токен ініціалізується, то усі об'єкти, які можуть бути зруйновані, повинні бути знищені (тобто, всі за винятком об'єктів, "що не підлягають знищенню", таких як ключі, що вбудовані у токені). Крім того, доступ звичайного користувача буде неможливим, доки SO не встановить PIN-код звичайного користувача. В залежності від токена, можуть бути створені деякі об'єкти "за замовчуванням", і атрибути деяких об'єктів можуть мати значення за замовчуванням.

Якщо токен має "захищений шлях аутентифікації", як зазначено у прапорі CKF_PROTECTED_AUTHENTICATION_PATH у CK_TOKEN_INFO, то це означає, що існує деякий спосіб для користувача для здійснення перевірки автентичності токена без необхідності відправлення застосуванням PIN через бібліотеку Cryptoki. Одна з таких можливостей полягає у тому, що користувач вводить PIN-код за допомогою PINpad (клавіатури для введення PIN клієнта) у сам токен або у пристрій слоту. Для ініціалізації токена таким захищеним шляхом аутентифікації необхідно, щоб параметр pPin у C_InitToken був NULL_PTR. Під час виконання C_InitToken, SO PIN буде введений через захищений шлях аутентифікації.

Якщо токен має захищений шлях аутентифікації, відмінний від PINpad, то він є залежним від токена, незалежно від того, чи використовується C_InitToken для ініціалізації токена.

Неможливо ініціалізувати токен, якщо Cryptoki виявляє, що будь-яке застосування має відкриту сесію з ним; коли виклик C_InitToken здійснюється за таких обставин, то виклик повертається з помилкою CKR_SESSION_EXISTS.

Функція C_InitToken не може бути достатньою, щоб належним чином ініціалізувати комплексні токени (complex token). У цій ситуації, механізм ініціалізації, що застосовується, виходить за рамки Cryptoki. Визначення "комплексного токена" є специфікою продукту.

Повернені значення:	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.

Приклад:

```
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) {
.
.
}

```

4.4.8. C_InitPIN

```

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

```

C_InitPIN ініціалізує PIN звичайного користувача.
hSession є дескриптором сесії;
pPin вказує на PIN-код звичайного користувача;
ulPinLen є довжиною PIN-коду в байтах.

Ця специфікація дозволяє значенню PIN-коду містити будь-які припустимі символи UTF8, але токен може накласти підмножину обмежень.

C_InitPIN може бути викликаний лише у стані «R/W SO Functions». Спроба викликати її з сеансу в будь-якому іншому стані повертається з помилкою CKR_USER_NOT_LOGGED_IN.

Якщо токен має "захищений шлях аутентифікації", як зазначено у прапорі CKF_PROTECTED_AUTHENTICATION_PATH у CK_TOKEN_INFO, то це означає, що існує деякий спосіб для користувача для здійснення перевірки автентичності токenu без необхідності відправлення застосуванням PIN через бібліотеку Cryptoki. Одна з таких можливостей полягає у тому, що користувач вводить PIN-код за допомогою PINpad (клавіатури для введення PIN клієнта) у сам токен або у пристрій слоту. Для ініціалізації токenu таким захищеним шляхом аутентифікації, необхідно щоб параметр pPin у C_InitPIN був NULL_PTR. Під час виконання C_InitPIN, у SO вводиться новий PIN-код через захищений шлях аутентифікації.

Якщо токен має захищений шлях аутентифікації, відмінний від PINpad (клавіатури для введення PIN клієнта), то він є залежним від токenu, незалежно від того, чи використовується C_InitPIN для ініціалізації доступу до токenu звичайного користувача.

Повернені значення: 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.

Приклад:

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

4.4.9. 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  
);
```

C_SetPIN змінює PIN-код користувача, який зараз є активним, або CKU_USER PIN, якщо сесія не є аутентифікованою (logged in).

hSession є дескриптором сесії;
pOldPin вказує на старий PIN-код;
ulOldLen є довжиною в байтах старого PIN-коду;
pNewPin вказує на новий PIN-код;
ulNewLen – це довжина в байтах нового PIN-коду.

Ця специфікація дозволяє значенню PIN-коду містити будь-які припустимі символи UTF8, але токен може накласти підмножину обмежень.

C_SetPIN може бути викликаний лише у станах "R/W Public Session", "R/W SO Functions", або «R/W User Functions». Спроба викликати її з сесії в будь-якому іншому стані видається з помилкою CKR_SESSION_READ_ONLY.

Якщо токен має "захищений шлях аутентифікації", як зазначено у прапорі CKF_PROTECTED_AUTHENTICATION_PATH у CK_TOKEN_INFO, то це означає, що існує деякий спосіб для користувача для здійснення перевірки автентичності токена без необхідності відправлення програмою PIN через бібліотеку Cryptoki. Одна з таких можливостей полягає у тому, що користувач вводить PIN-код за допомогою PINpad (клавіатури для введення PIN клієнта) у сам токен або у пристрій слоту. Щоб змінити поточний PIN-код користувача у токени при такому захищеному шляху аутентифікації, параметри pOldPin та pNewPin у C_SetPIN повинні бути NULL_PTR. Під час виконання C_SetPIN, поточний користувач повинен вводити старий PIN-код та новий PIN-код через захищений шлях аутентифікації. Не визначено, яким чином PINpad повинен використовуватися для введення двох PIN-кодів.

Якщо токен має захищений шлях аутентифікації, відмінний від PINpad, то він є залежним від токена, незалежно від того, чи може використовуватися C_SetPIN для модифікації PIN-коду поточного користувача .

Повернені значення: 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.

Приклад:

```
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) {  
    .  
    .  
}
```

4.5. Функції управління сесіями

Типова програма застосування може виконувати такі кроки для використання токена (зверніть увагу, що існують інші раціональні послідовності дій, які може виконувати програма):

1. Вибрати токен.

2. Зробити один або кілька викликів C_OpenSession для здійснення однієї або кількох сесій з токеном.

3. Викликати C_Login для реєстрації у токені користувача. Внаслідок того, що усі сесії між застосуванням та токеном мають спільний стан реєстрації, необхідно викликати C_Login тільки для однієї з сесій.

4. Виконати криптографічні операції за допомогою сесій з токеном.

5. Викликати C_CloseSession один раз для кожної сесії, яку застосування має з токеном, або викликати C_CloseAllSessions для закриття всіх сесій програми одночасно.

Програма може мати паралельні сесії з більш ніж одним токеном. Також можливо для токена, щоб мати паралельні сесії з більш ніж однією програмою.

Cryptoki надає наступні функції для управління сесіями:

4.5.1. 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  
);
```

C_OpenSession відкриває сесію між програмою та токеном у конкретному слоті.

slotID є ідентифікатором слоту;

flags вказує тип сесії;

pApplication є визначеним програмно покажчиком для передавання повідомлення зворотного виклику;

Notify є адресою функції повідомлення зворотного виклику;

phSession вказує на розміщення, де утримується дескриптор нової сесії.

Під час відкриття сесії з C_OpenSession, параметр flags складається із застосуванням логічного OR з нульовою або більшою кількістю бітів прапорів, визначених в типі даних CK_SESSION_INFO. З міркувань успадкування, біт CKF_SERIAL_SESSION повинен завжди бути встановленим; якщо виклик C_OpenSession не має цього встановленого біту, виклик не буде успішним та повинен повернути код помилки CKR_PARALLEL_NOT_SUPPORTED.

Можуть бути обмеження на кількість одночасних сеансів, які програма може мати з токеном, які можуть залежати від того, чи це сесія “read-only”, або “read/write”. При спробі відкрити сесію, яка неуспішна тому, що є занадто багато існуючих сесій певного типу, слід повернути CKR_SESSION_COUNT.

Якщо токен захищено від запису (як зазначено в структурі CK_TOKEN_INFO), тоді з ним може бути відкрито тільки сесії лише для читання.

Якщо програма, яка викликає C_OpenSession, вже має R/W SO сесію з токеном, то будь-яка спроба відкрити R/O сесію з токеном завершиться помилкою з кодом CKR_SESSION_READ_WRITE_SO_EXISTS.

Функція зворотного виклику Notify використовується Cryptoki для повідомлення програми про певні події. Якщо програма бажає підтримати зворотні виклики, вона повинна передати значення NULL_PTR як параметр Notify.

Повернені значення: 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.

Приклад: див C_CloseSession.

4.5.2. C_CloseSession

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

C_CloseSession закриває сесію між програмою застосування та токеном. hSession є дескриптором сесії.

Після закриття сеансу, всі об'єкти сесії, що були створені під час сеансу, автоматично знищуються, навіть якщо програма має інші сесії, "що використовують" ці об'єкти.

Якщо ця функція успішно викликана і вона закриває останню сесію між програмою та токеном, то стан аутентифікації/ логіна токена для програми повертається до публічної сесії. Отже, будь-які нові сесії для токена, що були відкриті, будуть або R/O Public , або R/W Public.

Залежно від токена, коли остання відкрита сесія між будь-яким застосуванням і токеном закрита, то токен може бути "вилучений" зі свого зчитувача (якщо існує така можливість).

Незважаючи на те, що ця C_CloseSession використовується для закриття сесії, повернене значення CKR_SESSION_CLOSED є помилкою повернення. Вона фактично вказує (можливо, малоімовірно) подію: коли виконувався виклик цієї функції, інший виклик C_CloseSession здійснив закриття даної сесії і цей

(інший) виклик завершив виконання першого. Таке використання сесій є поганою практикою і Cryptoki не може обіцяти, що буде відбуватися в цілому, якщо програма веде себе таким чином.

Повернені значення: 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.

Приклад:

```
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);
}
```

4.5.3. C_CloseAllSessions

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

C_CloseAllSessions закриває всі сесії між застосуванням та токеном.
slotID вказує ідентифікатор слоту токена.

Після закриття сесії всі об'єкти сесії, що були створені за цей сеанс, знищуються автоматично.

Після успішного виконання цієї функції стан логіна токена для застосування повертається до публічної сесії. Тобто, будь-які нові сесії токена, що будуть відкриті, будуть або R/O Public ,або R/W Public.

Залежно від токєну, коли останню відкриту сесію між будь-яким застосуванням і токєном закрито, токєн може бути "вилучєний" зі свого зчитувача (якщо існує така можливість).

Повернені значєння: 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.

Приклад:

```
CK_SLOT_ID slotID;  
CK_RV rv;  
  
.  
.  
rv = C_CloseAllSessions(slotID);
```

4.5.4. C_GetSessionInfo

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

C_GetSessionInfo отримує інформацію про сесію.
hSession є дескриптором сесії;
pInfo вказує на розташування, де утримується інформація про сесію.

Повернені значєння: 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.

Приклад:

```
CK_SESSION_HANDLE hSession;  
CK_SESSION_INFO info;  
CK_RV rv;  
  
.  
.  
rv = C_GetSessionInfo(hSession, &info);  
if (rv == CKR_OK) {
```

```

    if (info.state == CKS_RW_USER_FUNCTIONS) {
        .
        .
    }
    .
    .
}

```

4.5.5. C_GetOperationState

```

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

```

C_GetOperationState отримує копію стану криптографічних операцій сесії, що кодується як послідовність байтів.

hSession є дескриптор сесії;
 pOperationState вказує на місце, де утримується стан;
 pulOperationStateLen вказує на місце, де утримується довжина в байтах стану.

Хоча збережений стан, що виводиться C_GetOperationState, насправді не виробляється "криптографічним механізмом", проте C_GetOperationState використовує конвенцію, що описано в розділі 4.2.6 про виробництво вихідних даних.

Хоча "стан криптографічних операцій", який дана функція зберігає, варіюється від токена до токена, однак, цей стан є тим, що передбачено в якості вхідних даних для C_SetOperationState з метою відновити криптографічну діяльність сесії.

Розглянемо сесію, яка виконує операцію хешування повідомлення з використанням SHA-1 (тобто сесією використовується СКМ_SHA_1 механізм). Припустимо, що операція хешування повідомлення була ініціалізована належним чином, і що саме 80 байт даних були поставлені до цих пір в якості вхідних даних до SHA-1. Застосування тепер хоче "зберегти стан" цієї операції переварити так, щоб можна було відновити його пізніше. В даному конкретному випадку, так як SHA-1 обробляє 512 біт (64 байт) вхідних даних за один раз, то криптографічний стан операцій сесії, швидше за все, складається з трьох окремих частин: стан 160-бітної внутрішньої змінної зчеплення (chaining) SHA-1; 16 байт необроблених вхідних даних, а також деякі адміністративні дані, які вказують, що це збережений стан походить від сесії, яка виконує SHA-1 хешування. Узяті разом, цих трьох частин інформації достатньо, щоб продовжити поточну операцію хешування пізніше.

Розглянемо другу сесію, яка виконує операції шифрування з DES (блочний шифр з розміром блоку 64 біта) в CBC (шифр-блок зчеплення, cipher-block

chaining) режимі (тобто сесія використовує CKM_DES_CBC механізм). Припустимо, що саме 22 байт даних (на додаток до IV в режимі CBC) були надані до цих пір в якості вхідних даних до DES; це означає, що перші два 8-байтних блоки шифротексту вже були оброблені і виведені (у вихідні дані). У цьому випадку стан криптографічних операцій сесії, швидше за все, складається з трьох або чотирьох окремих частин: другий 8-байтовий блок зашифрованого тексту (це буде використовуватися у CBC для отримання наступного блоку зашифрованого тексту), 6 байт даних, які все ще очікують зашифрування, деякі адміністративні дані, які вказують, що це збережений стан виходить із сесії, яка виконує DES шифрування в режимі CBC, і, можливо, ключ DES, що використовується для шифрування (див. C_SetOperationState для отримання додаткової інформації про те треба чи ні присутність ключа в збереженому стані).

Якщо сесія виконує дві криптографічні операції одночасно (див. розділ 4.2.3), то стан криптографічної операції сесії буде містити всі необхідні відомості для відновлення обох операцій.

Спроба зберегти стан криптографічної операції сесії, яка на даний час не мала активні криптографічні операції (зашифрування, розшифрування, хешування, підписання без відновлення повідомлень, перевірка підпису без відновлення повідомлень, або будь-якого допустимого поєднання двох із них), повинна завершитися з помилкою CKR_OPERATION_NOT_INITIALIZED.

Спроба зберегти стан криптографічної операції сесії, яка виконує відповідну криптографічну операцію (або дві), але які не можуть бути виконані за будь-яких різних причин (деякі необхідні відомості про стан і / або ключова інформація, не можуть залишати токен, наприклад), повинні завершуватися з помилкою CKR_STATE_UNSAVEABLE.

Повертаються	значення:	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.		

Приклад: див C_SetOperationState.

4.5.6. 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  
);
```

C_SetOperationState відновлює стан криптографічних операцій сесії із рядку байтів, отриманих з C_GetOperationState.

hSession є дескриптор сесії;

pOperationState вказує на місце розташування збереженого стану;

ulOperationStateLen утримує довжину збереженого стану;

hEncryptionKey утримує дескриптор ключа, який буде використовуватися для поточних зашифрування або розшифрування операцій у відновленій сесії (або 0 якщо не потрібен ключ зашифрування чи розшифрування, або тому, що не така операція мається в збереженій сесії, або тому, що вся необхідна ключова інформації присутня в збереженому стані);

hAuthenticationKey утримує дескриптор ключа, який буде використовуватися для поточних операцій підписування, обчислення MAC, або перевіряння підпису у відновленій сесії (або 0, якщо не потрібен такий ключ, або тому, що не така операція мається в збереженій сесії, або тому, що вся необхідна ключова інформація присутня в збереженому стані).

Стан криптографічних операцій не обов'язково повинен бути отриманий з тієї ж сесії ("початкова сесії"), як і сесія, що в даний час відновлена ("сесія призначення"). Тим не менш, початкова сесія і сесія призначення повинні мати спільний стан сесії (наприклад, CKS_RW_USER_FUNCTIONS), і повинні бути із спільним токеном. Немає гарантії, що стан криптографічних операцій може бути перенесений через логіни, або між різними реалізаціями Cryptoki.

Якщо C_SetOperationState надається збережений стан криптографічних операцій, який він не може визначити, чи який не є допустимим збереженим станом (або стан криптографічних операцій з сесії з різним станом сесії, або стан криптографічних операцій з різних токенів), функція завершується з помилкою CKR_SAVED_STATE_INVALID.

Збережений стан, отриманий після виклику C_GetOperationState, може або не може містити інформацію про ключі, що використовуються для поточних операцій шифрування. Якщо збережений стан криптографічних операцій мав поточні операції зашифрування або розшифрування, а ключ, що використовується для роботи, не зберігається в стані, то він повинен бути представлений в C_SetOperationState як аргумент hEncryptionKey. Якщо це не так, то C_SetOperationState повинна повернути помилку CKR_KEY_NEEDED. Якщо ключ, що використовується для операцій, зберігається в стані, то він може поставлятися в hEncryptionKey аргументі, але це не вимагається.

Аналогічним чином, якщо збережений стан криптографічних операцій мав поточні операції підписування, обчислення MAC, або перевіряння підпису, а ключ, що використовувався для роботи, не зберігається в стані, то він повинен бути представлений в C_SetOperationState як аргумент hAuthenticationKey. Якщо це не так, то C_SetOperationState завершиться невдало з помилкою CKR_KEY_NEEDED. Якщо ключ, що використовується для операцій, зберігається в стані, то він може бути поставлений в hAuthenticationKey аргументі, але це не вимагається.

Якщо невідповідний ключ надається до C_SetOperationState виклику (наприклад, ненульовий дескриптор ключа надається у hEncryptionKey аргументі, але збережений стан криптографічної операції, що надається, не має поточних операцій зашифрування або розшифрування), то C_SetOperationState завершується з помилкою CKR_KEY_NOT_NEEDED.

Якщо ключ поставляється в якості аргументу C_SetOperationState і C_SetOperationState може якось визначити, що цей ключ не був ключем, що використовувався у початковій сесії для наданого стану криптографічних операцій (наприклад, це можна виявити, якщо ключ або хеш ключа знаходиться в збереженому стані), то C_SetOperationState завершується з помилкою CKR_KEY_CHANGED.

Застосування може розглянути CKF_RESTORE_KEY_NOT_NEEDED прапор у полі прапорів CK_TOKEN_INFO для того, щоб визначити, чи дійсно йому потрібно надати дескриптор ключа до C_SetOperationState дзвінки. Якщо цей прапор такий, то виклик C_SetOperationState ніколи не вимагає надання дескриптора ключа. Якщо цей прапор є хибним, то C_SetOperationState вимагає дескриптор ключа, і таким чином застосування повинно, ймовірно, завжди надавати дескриптори відповідних ключів при відновленні стану криптографічних операцій.

C_SetOperationState може успішно відновлювати стан криптографічних операцій для сесії, навіть якщо ця сесія має активні криптографічні операції або операції пошуку об'єкта на момент, коли C_SetOperationState викликається (поточні операції миттєво скасовуються).

Повертаються значення: 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.

Приклад:

```
CK_SESSION_HANDLE hSession;  
CK_MECHANISM digestMechanism;  
CK_ULONG ulStateLen;  
CK_BYTE data1[] = {0x01, 0x03, 0x05, 0x07};
```

```

CK_BYTE data2[] = {0x02, 0x04, 0x08};
CK_BYTE data3[] = {0x10, 0x0F, 0x0E, 0x0D, 0x0C};
CK_BYTE pDigest[20];
CK_ULONG ulDigestLen;
CK_RV rv;

.
.
/* Initialize hash operation */
rv = C_DigestInit(hSession, &digestMechanism);
assert(rv == CKR_OK);

/* Start hashing */
rv = C_DigestUpdate(hSession, data1, sizeof(data1));
assert(rv == CKR_OK);

/* Find out how big the state might be */
rv = C_GetOperationState(hSession, NULL_PTR, &ulStateLen);
assert(rv == CKR_OK);

/* Allocate some memory and then get the state */
pState = (CK_BYTE_PTR) malloc(ulStateLen);
rv = C_GetOperationState(hSession, pState, &ulStateLen);

/* Continue hashing */
rv = C_DigestUpdate(hSession, data2, sizeof(data2));
assert(rv == CKR_OK);

/* Restore state. No key handles needed */
rv = C_SetOperationState(hSession, pState, ulStateLen, 0, 0);
assert(rv == CKR_OK);

/* Continue hashing from where we saved state */
rv = C_DigestUpdate(hSession, data3, sizeof(data3));
assert(rv == CKR_OK);

/* Conclude hashing operation */
ulDigestLen = sizeof(pDigest);
rv = C_DigestFinal(hSession, pDigest, &ulDigestLen);
if (rv == CKR_OK) {
    /* pDigest[] now contains the hash of 0x01030507100F0E0D0C */
    .
    .
}

```

4.5.7. C_Login

```
CK_DEFINE_FUNCTION(CK_RV, C_Login)(  
    CK_SESSION_HANDLE hSession,  
    CK_USER_TYPE userType,  
    CK_UTF8CHAR_PTR pPin,  
    CK_ULONG ulPinLen  
);
```

За допомогою C_Login виконується реєстрація користувача у токені.
hSession є дескриптором сесії;
userType – це тип користувача;
pPin покажчик на PIN-код користувача;
ulPinLen є довжиною PIN-коду.

Ця специфікація дозволяє значенню PIN-коду містити будь-які припустимі символи UTF8, але токен може накласти підмножину обмежень.

Коли тип користувача є або CKU_SO, або CKU_USER за умови успішного виклику, то кожна із сесій програми буде перебувати у станах "R/W SO Functions", "R/W User Functions" або у стані "R/O User Functions". Якщо тип користувача буде CKU_CONTEXT_SPECIFIC, то поведінка функції C_Login залежить від контексту, в якому вона викликається. Неправильне використання цього типу користувача призведе до повернення значення CKR_OPERATION_NOT_INITIALIZED.

Якщо токен має "захищений шлях аутентифікації", як зазначено у прапорі CKF_PROTECTED_AUTHENTICATION_PATH у їх CK_TOKEN_INFO, то це означає, що існує деякий спосіб для користувача для здійснення перевірки автентичності токenu без необхідності відправлення програмою PIN через бібліотеку Ccryptoki. Одна з таких можливостей полягає у тому, що користувач вводить PIN-код за допомогою PINpad (клавіатури для введення PIN клієнта) у сам токен або у пристрій слоту. Або користувач може навіть не використовувати PIN, наприклад, аутентифікація може бути здійснена за допомогою деяких пристроїв читання відбитків пальців. Для входу в токен з захищеного шляху аутентифікації, параметр pPin у C_Login повинен бути NULL_PTR. Коли C_Login завершується, незалежно від того який метод аутентифікації, що підтримується токеном, буде виконуватися, то повернене значення CKR_OK означає, що користувач був успішно авторизований, а повернене значення CKR_PIN_INCORRECT означає, що користувачу було відмовлено в доступі.

Якщо є будь-які активні криптографічні операції або операції пошуку об'єкту в сесії програми, і при цьому C_Login успішно виконується цим застосуванням, то може або не може бути випадок, коли ці операції є все ще активними. Таким чином, перед реєстрацією (logging in), будь-які активні операції повинні бути завершені.

Якщо програма, що викликала C_Login, має відкриту з токеном R/O сесію, то вона буде не в змозі зареєструвати SO у сесії. Спроба зробити це, призведе до формування коду помилки CKR_SESSION_READ_ONLY_EXISTS.

C_Login можна викликати кілька разів без проміжних викликів C_Logout, якщо (і тільки якщо) існує ключ з атрибутом SCA_ALWAYS_AUTHENTICATE, встановленим у CK_TRUE, і користувачеві необхідно зробити криптографічну операцію за допомогою цього ключа. Див. далі розділ 3.7.8.

Повернені	значення:	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.		

Приклад: див. C_Logout.

4.5.8. C_Logout

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

За допомогою C_Logout користувач виходить з токена.
hSession є дескриптором сесії.

В залежності від поточного типу користувача, якщо виклик здійснено успішно, то кожна сесія програми буде у станах або "R/W Public Session", або "R/O Public Session".

Коли C_Logout успішно здійснено, то будь-який із дескрипторів на особисті об'єкти програмних стає недійсним (навіть, якщо користувач пізніше заново реєструється у токені, ці дескриптори залишаються недійсними). Крім того, усі особисті об'єкти сесії сесій, які належать до програми, знищуються.

Якщо є будь-які активні криптографічні операції або операції пошуку об'єкту в сесії програми, і при цьому C_Logout успішно виконується, то може або не може бути випадок, коли ці операції є все ще активними. Таким чином, перед виходом, будь-які активні операції повинні бути завершені.

Повернені значення: 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.

Приклад:

```
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) {
        .
        .
    }
}
```

4.6. Функції управління об'єктами

Cryptoki надає наступні функції управління об'єктами. Додаткові функції, що надаються спеціально для управління ключовими об'єктами, описані в розділі 4.3.

4.6.1. 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
);
```

C_CreateObject створює новий об'єкт.
hSession є дескриптором сесії;
pTemplate є покажчиком на шаблон об'єкта;
ulCount – кількість атрибутів у шаблоні;
phObject є покажчик на розташування, де утримується новий дескриптор об'єкту.

Якщо виклик `C_CreateObject` не підтримує точний шаблон, що надано йому, то він буде невдалим і повернеться без створення будь-якого об'єкта.

Якщо `C_CreateObject` використовуються для створення ключового об'єкта, то ключовий об'єкт буде мати атрибут `CKA_LOCAL`, встановлений у `CK_FALSE`. Якщо цей ключовий об'єкт є секретним або закритим ключем, то новий ключ буде мати атрибут `CKA_ALWAYS_SENSITIVE` із значенням `CK_FALSE` і атрибут `CKA_NEVER_EXTRACTABLE` із значенням `CK_FALSE`.

Тільки сесійні об'єкти можуть бути створені під час сесії «Тільки для читання». Тільки відкриті об'єкти можуть бути створені без реєстрації звичайного користувача.

Повернені	значення:	
<code>CKR_ATTRIBUTE_READ_ONLY</code> ,		<code>CKR_ARGUMENTS_BAD</code> ,
<code>CKR_ATTRIBUTE_VALUE_INVALID</code> ,		<code>CKR_ATTRIBUTE_TYPE_INVALID</code> ,
<code>CKR_DEVICE_ERROR</code> ,	<code>CKR_DEVICE_MEMORY</code> ,	<code>CKR_CRYPTOKI_NOT_INITIALIZED</code> ,
<code>CKR_DOMAIN_PARAMS_INVALID</code> ,		<code>CKR_DEVICE_REMOVED</code> ,
<code>CKR_GENERAL_ERROR</code> ,	<code>CKR_HOST_MEMORY</code> ,	<code>CKR_FUNCTION_FAILED</code> ,
<code>CKR_PIN_EXPIRED</code> ,		<code>CKR_OK</code> ,
<code>CKR_SESSION_HANDLE_INVALID</code> ,		<code>CKR_SESSION_CLOSED</code> ,
<code>CKR_TEMPLATE_INCOMPLETE</code> ,		<code>CKR_SESSION_READ_ONLY</code> ,
<code>CKR_TOKEN_WRITE_PROTECTED</code> ,	<code>CKR_USER_NOT_LOGGED_IN</code> .	

Приклад:

```
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) {
    .
    .
}

```

4.6.2. 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  
);
```

C_CopyObject копіює об'єкт, створюючи новий об'єкт.
hSession є дескриптором сесії;
hObject є дескриптором об'єкта;
pTemplate є покажчиком на шаблон для нового об'єкта;
ulCount – кількість атрибутів у шаблоні;
phNewObject є покажчиком на розташування, де утримується дескриптор копії об'єкта.

Шаблон може вказувати нові значення атрибутів об'єктів, які зазвичай можна змінити (наприклад, при копіюванні секретного ключа атрибут ключа CKA_EXTRACTABLE може змінюватися з CK_TRUE на CK_FALSE, але не навпаки. Якщо ця зміна відбувається, новий атрибут ключа CKA_NEVER_EXTRACTABLE матиме значення CK_FALSE. Аналогічним чином, шаблон може визначати, що новий атрибут ключа CKA_SENSITIVE буде мати значення CK_TRUE; новий ключ буде мати те ж саме значення атрибута CKA_ALWAYS_SENSITIVE, як і оригінальний ключ). Він може також визначати нові значення атрибутів CKA_TOKEN і CKA_PRIVATE (наприклад, щоб скопіювати об'єкт сесії у об'єкт токenu). Якщо шаблон визначає значення атрибута, який є несумісним з іншими існуючими атрибутами об'єкта, то виклик є невдалим з кодом повернення CKR_TEMPLATE_INCONSISTENT.

Якщо виклик C_CopyObject не підтримує точного шаблону, наданого йому, то він буде невдалим і повернеться без створення будь-якого об'єкта.

Тільки сесійні об'єкти можуть бути створені під час сесії «Тільки для читання». Тільки відкриті об'єкти можуть бути створені без реєстрації звичайного користувача.

Повернені	значення:	
CKR_ATTRIBUTE_READ_ONLY,		CKR_ARGUMENTS_BAD,
CKR_ATTRIBUTE_VALUE_INVALID,		CKR_ATTRIBUTE_TYPE_INVALID,
CKR_DEVICE_ERROR,	CKR_DEVICE_MEMORY,	CKR_CRYPTOKI_NOT_INITIALIZED,
CKR_FUNCTION_FAILED,	CKR_GENERAL_ERROR,	CKR_DEVICE_REMOVED,
CKR_OBJECT_HANDLE_INVALID,		CKR_HOST_MEMORY,
CKR_SESSION_CLOSED,		CKR_OK,
CKR_SESSION_READ_ONLY,		CKR_PIN_EXPIRED,
CKR_TOKEN_WRITE_PROTECTED,	CKR_USER_NOT_LOGGED_IN,	CKR_SESSION_HANDLE_INVALID,
		CKR_TEMPLATE_INCONSISTENT,

Приклад:

```
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);
    .
    .
}
```

4.6.3. C_DestroyObject

<pre>CK_DEFINE_FUNCTION(CK_RV, C_DestroyObject)(CK_SESSION_HANDLE hSession, CK_OBJECT_HANDLE hObject);</pre>
--

C_DestroyObject знищує об'єкт.
hSession є дескриптором сесії;
hObject є дескриптором об'єкта.

Тільки сесійні об'єкти можуть бути знищені протягом сесії, яка є сесією «Тільки для читання». Лише відкриті об'єкти можуть бути знищені, за винятком, коли у системі зареєстровано звичайного користувача.

Повернені значення: CKR_CRYPTOKI_NOT_INITIALIZED, KR_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.

Приклад: див. C_GetObjectSize

4.6.4. C_GetObjectSize

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

C_GetObjectSize отримує розмір об'єкта в байтах.

hSession є дескриптором сесії;

hObject є дескриптором об'єкта;

pulSize є покажчик на розміщення, де утримується розмір об'єкта у байтах.

Cryptoki не визначає, яке точне значення розміру об'єкта. Інтуїтивно, це деяка міра, скільки пам'яті токенау займає об'єкт. Якщо програма видаляє (скажімо) особисті об'єкти розміром S, то буде розумно припустити, що поле ulFreePrivateMemory структури CK_TOKEN_INFO токенау збільшується приблизно на S.

Повернені значення: 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

Приклад:

```
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);
    .
    .
}

```

4.6.5. C_GetAttributeValue

```

CK_DEFINE_FUNCTION(CK_RV, C_GetAttributeValue)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE hObject,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);

```

C_GetAttributeValue отримує значення одного або декількох атрибутів об'єкту.

hSession є дескриптором сесії;

hObject є дескриптором об'єкта;

pTemplate є покажчиком на шаблон, що визначає, яке значення атрибуту повинно бути отримано, і отримує значення атрибутів;

ulCount є числом атрибутів у шаблоні.

Для кожних (type, pValue, ulValueLen) трьох полів у шаблоні, C_GetAttributeValue виконує такий алгоритм:

1. Якщо вказано атрибут (наприклад, атрибут, визначений у полі type) для об'єкта, який неможливо відкрити (тому що об'єкт є чутливим/sensitive або таким, що не вилучається/ unextractable), то поле ulValueLen з цієї трійці модифікується для розміщення значення -1 (тобто, коли воно міняється на CK_LONG, воно містить -1).

2. Інакше, якщо вказаний атрибут для об'єкта є неприпустимим (об'єкт не має такого атрибута), то поле ulValueLen з цієї трійці модифікується для розміщення значення -1.

3. В іншому випадку, якщо поле pValue має значення NULL_PTR, то поле ulValueLen модифікується для утримання точної довжини вказаного атрибуту об'єкту.

4. Інакше, якщо довжина, що вказана в ulValueLen є достатньо великою для розміщення значення вказаного атрибуту об'єкта, то цей атрибут копіюється в буфер, який розташований в pValue і поле ulValueLen модифікується для розміщення точної довжини атрибута.

5. В іншому випадку поле ulValueLen модифікується для розміщення значення -1.

У випадку 1, що стосується будь-яких необхідних атрибутів, виклик повинен повернути значення CKR_ATTRIBUTE_SENSITIVE. У випадку 2, що стосується будь-яких потрібних атрибутів, виклик повинен повернути значення CKR_ATTRIBUTE_TYPE_INVALID. У випадку 5, що стосується будь-якого потрібного атрибуту, виклик повинен повернути значення CKR_BUFFER_TOO_SMALL. Зазвичай, якщо можна застосувати більш ніж один з цих кодів помилки, то Cryptoki може повернути будь-який з них. Тільки, якщо ні один з них не застосовується до будь-яких потрібних атрибутів, то буде повернуто CKR_OK.

У особливому випадку атрибут, значення якого являє собою масив атрибутів, наприклад CKA_WRAP_TEMPLATE, коли він передається з ненульовим значенням pValue, то якщо елементи pValue у масиві є NULL_PTR, тоді елементам ulValueLen масиву буде призначено потрібну довжину. Якщо елементи pValue масиву не є NULL_PTR, тоді елемент ulValueLen атрибутів повинен відображати простір, що відповідає покажчику pValue, і pValue заповнюється, якщо є достатньо місця. Тому важливо ініціалізувати вміст буферу перед викликом C_GetAttributeValue для отримання такого значення масиву. Якщо ulValueLen у масиві є не досить великим, то його буде встановлено у -1 і функція поверне CKR_BUFFER_TOO_SMALL; це робиться, якщо атрибут у аргументі pTemplate має занадто малий ulValueLen. Зверніть увагу, що будь-який атрибут, значення якого являє собою масив атрибутів, може бути коректно ідентифіковано за типом атрибута, що має встановлений біт CKF_ARRAY_ATTRIBUTE.

Зверніть увагу, що коди помилок CKR_ATTRIBUTE_SENSITIVE, CKR_ATTRIBUTE_TYPE_INVALID і CKR_BUFFER_TOO_SMALL не означають

дійсну помилку для C_GetAttributeValue. Якщо виклик C_GetAttributeValue повертає будь-яке з цих трьох значень, то виклику необхідно тим не менше обробити кожен атрибут шаблону для C_GetAttributeValue. Кожен атрибут у шаблоні, значення якого може бути повернуто при виклику C_GetAttributeValue, повернеться при виклику C_GetAttributeValue.

Повернені	значення:	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.		

Приклад:

```

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);
}

```

4.6.6. C_SetAttributeValue

```
CK_DEFINE_FUNCTION(CK_RV, C_SetAttributeValue)(
    CK_SESSION_HANDLE hSession,
    CK_OBJECT_HANDLE hObject,
    CK_ATTRIBUTE_PTR pTemplate,
    CK_ULONG ulCount
);
```

C_SetAttributeValue модифікує значення одного або декількох атрибутів об'єкту.

hSession є дескриптором сесії;
hObject є дескриптором об'єкта;
pTemplate є покажчиком на шаблон, що визначає, яке значення атрибуту повинно бути модифіковано та їх нові значення;
ulCount є числом атрибут у шаблоні.

Тільки сесійні об'єкти можуть бути змінені під час сесії «Тільки для читання».

Шаблон може визначати нові значення для будь-якого атрибуту об'єкта, які можуть модифікуватися. Якщо шаблон визначає значення атрибута, який є несумісним з іншими наявними атрибутами об'єкта, то виклик не виконується з кодом повернення CKR_TEMPLATE_INCONSISTENT.

Не всі атрибути можуть бути модифіковані; див розділ 2.7.7 для більш докладної інформації.

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

Приклад:

```
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) {
    .
    .
}

```

4.6.7. C_FindObjectsInit

```

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

```

C_FindObjectsInit ініціалізує пошук токена та об'єктів сесії, які задовольняють шаблону.

hSession є дескриптором сесії;

pTemplate є покажчиком на шаблон пошуку, що визначає значення відповідного(их) атрибута(ів);

ulCount – це кількість атрибутів у шаблоні пошуку.

Відповідний критерій – це точне співставлення байт за байтом з усіма атрибутами у шаблоні. Після знаходження всіх об'єктів, встановлюють ulCount у нульове значення.

Після виклику C_FindObjectsInit, програма може викликати C_FindObjects один або декілька разів для отримання дескрипторів об'єктів, які відповідають шаблону, і потім у кінцевому підсумку викликається C_FindObjectsFinal для завершення операції активного пошуку. В більшості випадків одна операція пошуку може бути активною в даний час для даної сесії.

Операція пошуку об'єкту буде шукати тільки об'єкти, які можна переглянути у сесії. Наприклад, пошук об'єкту в "R/W Public Session" не знайде будь-які особисті об'єкти (навіть, якщо один з атрибутів у шаблоні пошуку визначає, що пошук здійснюється для таких об'єктів).

Якщо операція пошуку є активною і об'єкти створюються або знищуються відповідно до шаблону пошуку для активної операції пошуку, то такі об'єкти можуть або не можуть бути знайдені за допомогою операції пошуку. Зверніть увагу, що це означає, що за цих обставин, операція пошуку може повернути неправильний дескриптор об'єкту.

Хоча C_FindObjectsInit може повернути значення CKR_ATTRIBUTE_TYPE_INVALID та CKR_ATTRIBUTE_VALUE_INVALID, це не обов'язково. Наприклад, якщо вона отримує шаблон пошуку з неіснуючими

атрибутами у ньому, то може повернути CKR_ATTRIBUTE_TYPE_INVALID або вона може ініціалізувати операцію пошуку, яка не буде порівнювати ніякі об'єкти, і поверне CKR_OK.

Повернені	значення:	
CKR_ATTRIBUTE_TYPE_INVALID,		CKR_ARGUMENTS_BAD,
CKR_CRYPTOKI_NOT_INITIALIZED,		CKR_ATTRIBUTE_VALUE_INVALID,
CKR_DEVICE_MEMORY,		CKR_DEVICE_ERROR,
CKR_FUNCTION_FAILED,	CKR_GENERAL_ERROR,	CKR_DEVICE_REMOVED,
CKR_OK,	CKR_OPERATION_ACTIVE,	CKR_HOST_MEMORY,
CKR_SESSION_CLOSED,	CKR_SESSION_HANDLE_INVALID.	CKR_PIN_EXPIRED,

Приклад: див. C_FindObjectsFinal.

4.6.8. 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  
);
```

C_FindObjects продовжує пошук тдля окену та об'єктів сесії, які задовольняють шаблону, для отримання дескрипторів додаткових об'єктів.

hSession є дескриптором сесії;

phObject є покажчиком на розташування, де утримується список (масив) дескрипторів додаткових об'єктів;

ulMaxObjectCount є максимальним числом дескрипторів об'єктів, які будуть повернуті;

pulObjectCount є покажчиком на розташування, де утримується фактична кількість повернених дескрипторів об'єктів.

Якщо більш немає об'єктів, що відповідають шаблону, то розташування, на яке вказує pulObjectCount, отримує значення 0.

Пошук повинен бути ініціалізовано за допомогою C_FindObjectsInit

Повернені	значення:	
CKR_CRYPTOKI_NOT_INITIALIZED,		CKR_ARGUMENTS_BAD,
CKR_DEVICE_MEMORY,		CKR_DEVICE_ERROR,
CKR_FUNCTION_FAILED,	CKR_GENERAL_ERROR,	CKR_DEVICE_REMOVED,
CKR_OK,	CKR_OPERATION_NOT_INITIALIZED,	CKR_HOST_MEMORY,
CKR_SESSION_HANDLE_INVALID.		CKR_SESSION_CLOSED,

Приклад: див. C_FindObjectsFinal

4.6.9. C_FindObjectsFinal

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

C_FindObjectsFinal припиняє пошук для токена та сесійних об'єктів.
hSession є дескриптором сесії.

Повернені значення: 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.

Приклад:

```
CK_SESSION_HANDLE hSession;  
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);
```

4.7. Функції зашифрування

Cryptoki надає такі функції для зашифрування даних:

4.7.1. C_EncryptInit

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

C_EncryptInit ініціалізує операцію шифрування.

hSession є дескриптором сесії;

pMechanism є покажчиком на механізм шифрування;

hKey є дескриптором ключа шифрування.

Атрибут CKA_ENCRYPT ключа шифрування, який вказує на те, чи підтримує ключ операцію зашифрування, повинен бути із значенням CK_TRUE.

Після виклику C_EncryptInit програма може або викликати функцію C_Encrypt для зашифрування однієї частини даних, та ні разу не викликати C_EncryptUpdate, або викликати C_EncryptUpdate функцію декілька разів для зашифрування декількох частин даних, після чого слідує функція C_EncryptFinal.

Операція зашифрування є активною, поки програма не використає новий виклик C_Encrypt, або виклик C_EncryptFinal для реального отримання остаточної порції зашифрованого тексту. Для обробки додаткових даних (однієї або декількох частин) програма знову повинна викликати C_EncryptInit.

Повернені значення: 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.

Приклад: див C_EncryptFinal.

4.7.2. 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  
);
```

C_Encrypt зашифровує одну частину даних.
hSession є дескриптором сесії;
pData є покажчиком на дані;
ulDataLen є довжиною даних в байтах;
pEncryptedData є покажчиком на розташування, де утримуються зашифровані дані;
pulEncryptedDataLen є покажчиком на розташування, де утримується довжина зашифрованих даних в байтах.

C_Encrypt використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція зашифрування повинна бути ініційована за допомогою C_EncryptInit. Виклик C_Encrypt завжди припиняє активну операцію зашифрування, окрім випадку, коли він повертає CKR_BUFFER_TOO_SMALL або виклик є успішним (тобто, повертається CKR_OK) для визначення довжини буфера, необхідного для утримання зашифрованого тексту (див. Конвенцію в розділі 4.2.6).

C_Encrypt не може бути використана для припинення мультичастинних операцій і повинна викликатися після C_EncryptInit без проміжних викликів C_EncryptUpdate.

Для деяких механізмів зашифрування даних, введення незашифрованих текстових даних має певні обмеження довжини (оскільки механізм може зашифрувати лише відносно короткі частини тексту, або тому, що введення даних для цього механізму повинно складатися з певної/кратної кількості блоків). Якщо ці обмеження не виконуються, то C_Encrypt не виконується і код повернення буде CKR_DATA_LEN_RANGE.

Звичайний текст і зашифрований текст можуть знаходитися в одному і тому ж місці, тобто, це є вірно, якщо pData і pEncryptedData вказують на одне і теж місце розташування.

Для більшості механізмів C_Encrypt є еквівалентом послідовності операцій C_EncryptUpdate, за якими слідує C_EncryptFinal.

Повернені значення:	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_OPERATION_NOT_INITIALIZED,	CKR_OK,
CKR_SESSION_HANDLE_INVALID.	CKR_SESSION_CLOSED,

Приклад: див C_EncryptFinal для прикладу подібних функцій.

4.7.3. 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
);
```

C_EncryptUpdate продовжує мультичастинну операцію зашифрування, та обробку іншої частини даних.

hSession є дескриптором сесії;

pPart є покажчик на частину даних;

ulPartLen є довжиною частини даних;

pEncryptedPart є покажчик на розташування, де утримується частина зашифрованих даних;

pulEncryptedPartLen є покажчик на розташування, де утримується довжина частини зашифрованих даних в байтах.

C_EncryptUpdate використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операцію зашифрування повинна бути ініційовано за допомогою C_EncryptInit. Функція C_EncryptUpdate може бути викликана будь-яку кількість разів. Виклик C_EncryptUpdate, що призводить до помилки, яка є відмінною від CKR_BUFFER_TOO_SMALL, припиняє поточну операцію зашифрування.

Звичайний текст і зашифрований текст можуть знаходитися в одному і тому ж місці, тобто, це є вірним, що pPart і pEncryptedPart вказують на одне і те ж місце розташування.

Повернені	значення:	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.

Приклад: див C_EncryptFinal.

4.7.4. C_EncryptFinal

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

C_EncryptFinal завершує мультичастинну операцію зашифрування.
hSession є дескриптором сесії;
pLastEncryptedPart є покажчиком на розташування, де утримується остання частина зашифрованих даних, якщо така є;
pulLastEncryptedPartLen є покажчиком на розташування, де утримується довжина останньої частини зашифрованих даних.

C_EncryptFinal використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція зашифрування повинно бути ініційована за допомогою C_EncryptInit. Виклик C_EncryptFinal завжди припиняє активну операцію зашифрування, окрім випадку, коли він повертає CKR_BUFFER_TOO_SMALL або виклик є успішним (тобто, повертається CKR_OK) для визначення довжини буфера, необхідного для утримання зашифрованого тексту (див. Конвенцію, розділ 4.2.6).

Для деяких мультичастинних механізмів шифрування, введення незашифрованих текстових даних має певні обмеження довжини, тому що механізм введення даних повинен складатися з певної/кратної кількості блоків. Якщо ці обмеження не виконуються, то C_EncryptFinal не виконується, а код повернення буде CKR_DATA_LEN_RANGE.

Повернені значення: 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.

Приклад:

```

#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_DES_CBC_PAD, 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) {
    .
    .
}
}

```

4.8. Функції розшифрування

Cryptoki надає такі функції для розшифрування даних:

C_DecryptInit

```

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

```

C_DecryptInit ініціалізує операції розшифрування.

hSession є дескриптором сесії;

pMechanism є покажчиком на механізм розшифрування;

hKey є дескриптором ключа дешифрування.

Атрибут СКА_DECRYPT ключа розшифрування, який вказує, чи ключ підтримує розшифрування, повинен бути із значенням CK_TRUE.

Після виклику C_DecryptInit, застосування може або викликати функцію C_Decrypt для розшифрування однієї частини даних та ні разу не викликати C_DecryptUpdate, або викликати цю функцію C_DecryptUpdate декілька разів для розшифрування декількох частин даних, після чого слідує функція C_DecryptFinal. Операція розшифрування є активною, поки програма не використає виклик C_Decrypt або C_DecryptFinal для реального отримання остаточної порції зашифрованого тексту. Для обробки додаткових даних (однієї або декількох частин), програма знову повинна викликати C_DecryptInit .

Повернені значення:

CKR_CRYPTOKI_NOT_INITIALIZED,
 CKR_DEVICE_MEMORY,
 CKR_FUNCTION_CANCELED,
 CKR_GENERAL_ERROR,

CKR_ARGUMENTS_BAD,
 CKR_DEVICE_ERROR,
 CKR_DEVICE_REMOVED,
 CKR_FUNCTION_FAILED,
 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.

Приклад: див. C_DecryptFinal

4.8.1. 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  
);
```

C_Decrypt виконує розшифрування зашифрованих даних.
hSession є дескриптором сесії;
pEncryptedData є покажчиком на зашифровані дані;
ulEncryptedDataLen – це довжина зашифрованих даних;
pData вказує є покажчиком на розташування, де утримуються відновлені дані;
pulDataLen є покажчиком на розташування, де утримуються довжина відновлених даних.

C_DecryptUpdate використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція розшифрування повинна була ініціалізована за допомогою C_DecryptInit. Виклик C_Decrypt завжди припиняє активну операцію розшифрування, за винятком випадку, коли вона повертає CKR_BUFFER_TOO_SMALL або виклик є успішним (тобто, повертається CKR_OK) та визначає довжину буфера, необхідного для розміщення звичайного тексту (див. Конвенцію, розділ 4.2.6).

C_Decrypt не може бути використана для припинення багато-поточної операції, та повинна бути викликана після C_DecryptInit без проміжних викликів функції C_DecryptUpdate.

Зашифрований текст і звичайний текст можуть знаходитися в одному місці, тобто, це є вірним, якщо pEncryptedData і pData визначають одне й те саме місце.

Якщо неможливо розшифрувати дані введеного зашифрованого тексту, тому що довжина є неналежною, то будь-яке із значень

CKR_ENCRYPTED_DATA_INVALID

або

CKR_ENCRYPTED_DATA_LEN_RANGE можуть бути повернені.

Для більшості механізмів C_Decrypt є еквівалентом послідовності операцій C_DecryptUpdate, після чого слідує C_DecryptFinal.

Повернені	значення:	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,	

Приклад: Див C_DecryptFinal

4.8.2. 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  
);
```

C_DecryptUpdate продовжує мультичастинну операцію розшифрування та обробку іншої частини зашифрованих даних.

hSession є дескриптором сесії;

pEncryptedPart є покажчиком на частину зашифрованих даних;

ulEncryptedPartLen є довжиною частини зашифрованих даних;

pPart є покажчиком на розташування, де утримується частина розшифрованих даних;

pulPartLen є покажчиком на розташування, де утримується довжина частини розшифрованих даних.

C_DecryptUpdate використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція розшифрування повинна бути ініціалізована за допомогою функції C_DecryptInit. Ця функція може визиватися будь-яку кількість разів поспіль. Виклик C_DecryptUpdate, що призводить до помилки, яка є відмінною від CKR_BUFFER_TOO_SMALL, припиняє поточну операцію розшифрування.

Зашифрований текст і звичайний текст можуть знаходитися в одному і тому ж місці, тобто, це є вірним, якщо pEncryptedPart і pPart вказують на одне і те ж місце розташування.

Повернені	значення:	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.	

Приклад: Див C_DecryptFinal

4.8.3. C_DecryptFinal

```
CK_DEFINE_FUNCTION(CK_RV, C_DecryptFinal)(  
    CK_SESSION_HANDLE hSession,  
    CK_BYTE_PTR pLastPart,  
    CK_ULONG_PTR pulLastPartLen  
);
```

C_DecryptFinal завершує багаточастинну операцію розшифрування. hSession є дескриптором сесії; pLastPart є покажчиком на розташування, де утримується остання частина розшифрованих даних, якщо є; pulLastPartLen є покажчиком на розташування, де утримується довжина останньої частини відновлених даних.

C_DecryptFinal використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція розшифрування повинна бути ініціалізована за допомогою функції C_DecryptInit. Виклик C_DecryptFinal завжди припиняє активну операцію розшифрування, окрім випадку, коли він повертає CKR_BUFFER_TOO_SMALL, або виклик є успішним (тобто, повертається CKR_OK) для визначення довжини буфера, необхідного для утримання звичайного тексту (див. Конвенцію, розділ 4.2.6).

Якщо неможливо розшифрувати дані введеного зашифрованого тексту, тому що довжина є неналежною, то будь-яке із значень CKR_ENCRYPTED_DATA_INVALID або CKR_ENCRYPTED_DATA_LEN_RANGE може бути повернено.

Повернені значення: 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.

Приклад:

```
#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_DES_CBC_PAD, 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) {
    .
    .
}
}

```

4.9. Функції хешування повідомлення

Cryptoki надає наступні функції для хешування (дайджесту - digesting) даних:

4.9.1. C_DigestInit

```

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

```

C_DigestInit ініціалізує операцію хешування повідомлення.
hSession є дескриптором сесії;
pMechanism є покажчик на механізм хешування.

Після виклику функції C_DigestInit застосування може або викликати C_Digest для обчислення хеш значення даних однієї частини; та ні разу не викликати C_DigestUpdate, або викликати функцію C_DigestUpdate декілька разів для хешування даних декількох частин, після чого слідує функція C_DigestFinal. Операція хешування даних є активною, поки програма не використовує виклик C_Digest або C_DigestFinal для реального отримання хеш значення повідомлення

(див. Конвенцію, розділ 4.2.6). Для обробки додаткових даних (однієї або декількох частин), програма знову повинна викликати C_DigestInit.

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

Приклад: див C_DigestFinal.

◆ 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  
);
```

C_Digest здійснює обчислення хеш даних однієї частини.
hSession є дескриптором сесії;
pData є покажчик на дані;
ulDataLen є довжиною даних;
pDigest є покажчик на розташування, де утримується хеш значення повідомлення;
pulDigestLen є покажчик на розташування, де утримується довжина хеш значення повідомлення.

C_Digest використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція дайджесту повинна бути ініційована за допомогою C_DigestInit. Виклик C_Digest завжди припиняє активну операцію дайджесту, крім випадку, коли він повертає CKR_BUFFER_TOO_SMALL або виклик є успішним (тобто, повертається CKR_OK) для визначення довжини буфера, необхідного для утримання хеш значення повідомлення (див. Конвенцію, розділ 4.2.6).

C_Digest не може бути використана для припинення мультичастинних операцій і повинна викликатися після C_DigestInit без проміжних викликів C_DigestUpdate.

Введенні дані і виведене хеш значення можуть бути в одному і тому ж місці розташування, тобто, це є вірним, якщо pData і pDigest вказують на одне і теж місце розташування.

C_Digest є еквівалентом послідовності операцій C_DigestUpdate, за якими слідує C_DigestFinal.

Повернені	значення:	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.		

Приклад: див C_DigestFinal.

4.9.2. C_DigestUpdate

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

C_DigestUpdate продовжує мультичастинну операцію хешування повідомлення та обробку іншої частини даних.

hSession є дескриптором сесії,

pPart є покажчиком частину даних;

ulPartLen є довжиною частини даних.

Операція дайджесту повідомлення повинна бути ініційована за допомогою C_DigestInit. Виклик цієї функції може бути здійснено будь-яку кількість разів. Виклик C_DigestUpdate, що призводить до помилки, припиняє поточну операцію хешування.

Повернені	значення:	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.		

Приклад: див C_DigestFinal

4.9.3. C_DigestFinal

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

C_DigestFinal завершує мультичастинну операцію хешування повідомлення, та повертає хеш значення повідомлення.

hSession є дескриптором сесії;

pDigest є покажчик на розташування, де утримується хеш значення повідомлення;

pulDigestLen є покажчик на розташування, де утримується довжина хеш значення повідомлення.

C_DigestFinal використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція хешування повинна бути ініційована за допомогою C_DigestInit. Виклик C_DigestFinal завжди припиняє активну операцію шифрування, окрім випадку, коли він повертає CKR_BUFFER_TOO_SMALL або виклик є успішним (тобто, повертається CKR_OK) для визначення довжини буфера, необхідного для утримання хеш значення повідомлення (див. Конвенцію, розділ 4.2.6).

Повернені	значення:	
CKR_BUFFER_TOO_SMALL,		CKR_ARGUMENTS_BAD,
CKR_DEVICE_ERROR,	CKR_DEVICE_MEMORY,	CKR_CRYPTOKI_NOT_INITIALIZED,
CKR_FUNCTION_CANCELED,		CKR_DEVICE_REMOVED,
CKR_GENERAL_ERROR,	CKR_HOST_MEMORY,	CKR_FUNCTION_FAILED,
CKR_OPERATION_NOT_INITIALIZED,		CKR_OK,
CKR_SESSION_HANDLE_INVALID.		CKR_SESSION_CLOSED,

Приклад:

```
CK_SESSION_HANDLE hSession;  
CK_MECHANISM mechanism = {  
    CKM_MD5, NULL_PTR, 0  
};  
CK_BYTE data[] = {...};  
CK_BYTE digest[16];  
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);

```

4.10. Функції обчислення підпису та MAC

Cryptoki повинен надавати такі функції для підписування даних (для цілей Cryptoki ці операції також охоплюють коди аутентифікації повідомлення - MAC, Message Authentication Code):

4.10.1. C_SignInit

```

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

```

C_SignInit ініціалізує операцію підписування, де підпис додається до даних.
hSession є дескриптором сесії;
pMechanism є покажчик на механізм підпису;
hKey є дескриптор ключа підпису.

Атрибут CKA_SIGN ключа підпису, що вказує, чи підтримує ключ підписування, повинен бути встановлений у CK_TRUE.

Після виклику функції C_SignInit, застосування може або викликати C_Sign для підпису однієї частини; та ні разу не викликати C_SignUpdate або викликати функцію C_SignUpdate декілька разів для підписування даних декількох частин, після чого слідує функція C_SignFinal. Операція підписування є активною, поки програма не використає виклик C_Sign або C_SignFinal для реального отримання значення підпису. Для обробки додаткових даних (однієї або декількох частин) програма знову повинна викликати функцію C_SignInit.

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

Приклад: див C_Sign.

4.10.2. 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  
);
```

C_Sign призначена для підписування однієї частини даних, де підпис додається даних.

hSession є дескриптором сесії;
pData є покажчик на данні;
ulDataLen є довжиною даних;
pSignature є покажчик на розташування, де утримується підпис;
pulSignatureLen є покажчик на розташування, де утримується довжина підпису.

C_Sign використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція підписування повинна бути ініційована за допомогою C_SignInit. Виклик C_Sign завжди припиняє активну операцію підписування, окрім випадку, коли він повертає CKR_BUFFER_TOO_SMALL або виклик є успішним (тобто, повертається CKR_OK) для визначення довжини буфера, необхідного для утримання підпису (див. Конвенцію, розділ 4.2.6).

C_Sign не може бути використана для припинення мультичастинних операцій і повинна викликатися після C_SignInit без проміжних викликів функції C_SignUpdate.

Для більшості механізмів, C_Sign є еквівалентом послідовності операцій C_SignUpdate, за якими слідує C_SignFinal.

Повернені	значення:	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.		

Приклад: див. C_SignFinal

4.10.3. C_SignUpdate

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

C_SignUpdate продовжує багаточастинну операцію підписування, оброблюючи іншу частину даних.

hSession є дескриптор сесії,

pPart є покажчик на частину даних;

ulPartLen є довжиною частини даних.

Операція підписування повинна бути ініціалізована за допомогою функції C_SignInit. Ця функція може бути викликана будь-яку кількість разів поспіль. Виклик C_SignUpdate, що призводить до помилки, припиняє поточну операцію підписування.

Повернені	значення:	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.		

Приклад: див C_SignFinal.

4.10.4. C_SignFinal

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

C_SignFinal завершує багато частинну операцію підписування поверненням значення підпису.

hSession є дескриптором сесії;
 pSignature є покажчик на розташування, де утримується значення підпису;
 pulSignatureLen є покажчик на розташування, де утримується довжина підпису.

C_SignFinal використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Операція підписування повинна бути ініціалізована за допомогою функції C_SignInit. Виклик C_SignFinal завжди припиняє активну операцію підписування, за винятком, коли вона повертає CKR_BUFFER_TOO_SMALL або виклик є успішним (тобто, повертається значення CKR_OK) з визначенням довжини буфера, необхідно для утримання значення підпису (див. Конвенцію, розділ 4.2.6).

Повернені	значення:	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.		

Приклад:

```

CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_MECHANISM mechanism = {
    CKM_DES_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);
    .
    .
}

```

4.11. Функції для перевіряння підпису та MAC

Cryptoki повинно надавати такі функції для перевіряння підпису даних (для цілей Cryptoki, ці операції також охоплюють коди аутентифікації повідомлення – MAC, Message Authentication Code):

4.11.1. C_VerifyInit

```

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

```

C_VerifyInit ініціалізує операції перевіряння, де підпис додається до даних. hSession є дескриптором сесії; pMechanism є покажчик на структуру, що визначає механізм перевіряння; hKey є дескриптором ключа перевіряння.

Атрибут CKA_VERIFY ключа перевіряння, що вказує, чи підтримує ключ перевіряння підпису, повинен бути встановлений у CK_TRUE.

Після виклику функції C_VerifyInit, застосування може або викликати C_Verify для перевіряння підпису даних однієї частини; та ні разу не викликати C_VerifyUpdate, або викликати функцію C_VerifyUpdate декілька разів для перевіряння підпису даних декількох частин, після чого слідує функція C_VerifyFinal. Операція перевіряння є активною, поки програма не викликає C_Verify або C_VerifyFinal. Для обробки додаткових даних (однієї або декількох частин) програма знову повинна викликати функцію C_VerifyInit.

Повернені	значення:	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.		

Приклад: див C_Verify

4.11.2. 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
);
```

C_Verify перевіряє підпис за один прохід операції (single-part operation), де підпис додається до даних.

hSession є дескриптором сесії;

pData є покажчик на данні, що підписані;

ulDataLen є довжиною даних;

pSignature є покажчик на значення підпису;

ulSignatureLen є довжиною підпису.

Операція перевіряння повинна бути ініційована за допомогою C_VerifyInit. Виклик C_Verify завжди припиняє активну операцію перевіряння.

Успішний виклик C_Verify повинен повертати або значення CKR_OK (з вказівкою, який існуючий підпис є дійсним/ не хибним), або

CKR_SIGNATURE_INVALID (з вказівкою, який існуючий підпис є недійсний/хибний). Якщо підпис визнається недійсним тільки через його довжину, то повинно бути повернуто CKR_SIGNATURE_LEN_RANGE. У будь-якому разі, активна операція підписування припиняється.

C_Verify не може бути використана для припинення мультичастинних операцій і повинна викликатися після C_VerifyInit без проміжних викликів функції C_VerifyUpdate.

Для більшості механізмів, C_Verify є еквівалентом послідовності операцій C_VerifyUpdate, за якими слідує C_VerifyFinal.

Повернені	значення:	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.		

Приклад: див. C_VerifyFinal.

4.11.3. C_VerifyUpdate

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

C_VerifyUpdate продовжує мультичастинну операцію перевіряння, оброблюючи інші частини даних.

hSession є дескриптором сесії,

pPart є покажчик на частину даних;

ulPartLen є довжиною частини даних.

Операція перевіряння повинна бути ініційована за допомогою C_VerifyInit. Ця функція може бути викликана будь-яку кількість разів поспіль. Виклик C_VerifyUpdate, що призводить до помилки, припиняє поточну операцію перевіряння.

Повернені	значення:	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_HANDLE_INVALID.

CKR_SESSION_CLOSED,

Приклад: див C_VerifyFinal.

4.11.4. C_VerifyFinal

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

C_VerifyFinal завершує мультичастинну операцію перевірення підпису.
hSession є дескриптором сесії;
pSignature є покажчик на значення підпису;
ulSignatureLen є довжиною підпису.

Операція перевірення повинна бути ініційована за допомогою функції C_VerifyInit. Виклик C_VerifyFinal завжди припиняє активну операцію перевірення.

При успішному виклику C_VerifyFinal повинно бути повернене або значення CKR_OK (із зазначенням того, що підпис, що застосовується, є чинним/ не хибним) або CKR_SIGNATURE_INVALID (із зазначенням того, що підпис, що використовується, є недійсним/ хибним). Якщо можна вважати підпис недійсним на основі його довжини, то повинно повертатися значення CKR_SIGNATURE_LEN_RANGE. У будь-якому з цих випадків, активна операція перевірення припиняється.

Повернені значення:	
CKR_CRYPTOKI_NOT_INITIALIZED,	CKR_ARGUMENTS_BAD,
CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY,	CKR_DATA_LEN_RANGE,
CKR_FUNCTION_CANCELED,	CKR_DEVICE_REMOVED,
CKR_GENERAL_ERROR,	CKR_FUNCTION_FAILED,
CKR_HOST_MEMORY,	CKR_OK,
CKR_OPERATION_NOT_INITIALIZED,	CKR_SESSION_CLOSED,
CKR_SESSION_HANDLE_INVALID,	CKR_SIGNATURE_INVALID,
CKR_SIGNATURE_LEN_RANGE.	

Приклад:

```
CK_SESSION_HANDLE hSession;  
CK_OBJECT_HANDLE hKey;  
CK_MECHANISM mechanism = {  
    CKM_DES_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));
    .
    .
}

```

4.12. Функції керування ключами

Cryptoki повинна надавати такі функції для управління ключами:

4.12.1. 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
);

```

C_GenerateKey генерує секретний ключ при створенні нового об'єкту. hSession є дескриптором сесії; pMechanism є покажчик на механізм генерації; pTemplate є покажчик на шаблон нового ключа; ulCount – це кількість атрибутів у шаблоні; phKey є покажчик на розташування, де утримується дескриптор нового ключа.

Атрибут CKA_CLASS повинен мати значення CKO_SECRET_KEY.

Оскільки тип ключа, що генерується, є неявним в механізмі генерації, то у шаблоні не потрібно застосовувати тип ключа. Якщо вказати тип ключа, який є несумісним з механізмом генерації, то C_GenerateKey не виконується і повертається код помилки CKR_TEMPLATE_INCONSISTENT. Атрибут CKA_CLASS оброблюється подібним чином.

Якщо виклик `C_GenerateKey` не підтримує визначеного шаблону, то він не виконується і повертається без створення об'єкта.

Об'єкт, створений при успішному виклиці `C_GenerateKey`, буде мати атрибут `CKA_LOCAL` зі значенням `CK_TRUE` (за замовчуванням).

Повернені	значення:
<code>CKR_ATTRIBUTE_READ_ONLY,</code>	<code>CKR_ARGUMENTS_BAD,</code>
<code>CKR_ATTRIBUTE_VALUE_INVALID,</code>	<code>CKR_ATTRIBUTE_TYPE_INVALID,</code>
<code>CKR_DEVICE_ERROR,</code>	<code>CKR_CRYPTOKI_NOT_INITIALIZED,</code>
<code>CKR_DEVICE_MEMORY,</code>	<code>CKR_DEVICE_REMOVED,</code>
<code>CKR_FUNCTION_CANCELED,</code>	<code>CKR_FUNCTION_FAILED,</code>
<code>CKR_GENERAL_ERROR,</code>	<code>CKR_HOST_MEMORY,</code>
<code>CKR_MECHANISM_INVALID,</code>	<code>CKR_MECHANISM_PARAM_INVALID,</code>
<code>CKR_OK,</code>	<code>CKR_OPERATION_ACTIVE,</code>
<code>CKR_SESSION_CLOSED,</code>	<code>CKR_PIN_EXPIRED,</code>
<code>CKR_SESSION_READ_ONLY,</code>	<code>CKR_SESSION_HANDLE_INVALID,</code>
<code>CKR_TEMPLATE_INCONSISTENT,</code>	<code>CKR_TEMPLATE_INCOMPLETE,</code>
<code>CKR_USER_NOT_LOGGED_IN.</code>	<code>CKR_TOKEN_WRITE_PROTECTED,</code>

Приклад:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_MECHANISM mechanism = {
    CKM_DES_KEY_GEN, NULL_PTR, 0
};
CK_RV rv;

.
.
rv = C_GenerateKey(hSession, &mechanism, NULL_PTR, 0, &hKey);
if (rv == CKR_OK) {
    .
    .
}
```

4.12.2. 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  
);
```

C_GenerateKeyPair генерує пару ключів відкритий/ особистий, під час створення нових ключових об'єктів.

hSession є дескриптором сесії;
pMechanism є показчик на механізм генерації;
pPublicKeyTemplate є показчик на шаблон відкритого ключа;
ulPublicKeyAttributeCount – це кількість атрибутів у шаблоні відкритого ключа;
pPrivateKeyTemplate є показчик на шаблон для особистого ключа;
ulPrivateKeyAttributeCount – це кількість атрибутів у шаблоні особистого ключа;
phPublicKey є показчик на розташування, де утримується дескриптор нового відкритого ключа;
phPrivateKey є показчик на розташування, де утримується дескриптор нового особистого ключа.

Оскільки типи ключів, що генеруються, є неявними у механізмі генерації ключової пари, то у шаблоні не потрібно застосовувати типи ключа. Якщо один із шаблонів застосовує тип ключа, який є несумісним з механізмом генерації, то C_GenerateKeyPair не виконується і повертається код помилки CKR_TEMPLATE_INCONSISTENT. Атрибут CKA_CLASS оброблюється подібним чином.

Якщо виклик C_GenerateKeyPair не підтримує шаблон, то він не виконується і повертається без створення ключових об'єктів.

Виклик C_GenerateKeyPair ніколи не створює тільки один ключ. Виклик не виконується і не створюється ключ; або він може бути успішним і створити відповідну ключову пару: відкритий/особистий.

Ключові об'єкти, створені успішно при виклику C_GenerateKeyPair, будуть мати атрибут CKA_LOCAL із значеннями CK_TRUE.

Повернені	значення:	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.

Приклад:

```

CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hPublicKey, hPrivateKey;
CK_MECHANISM mechanism = {
    CKM_RSA_PKCS_KEY_PAIR_GEN, NULL_PTR, 0
};
CK_ULONG modulusBits = 768;
CK_BYTE publicExponent[] = { 3 };
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_MODULUS_BITS, &modulusBits, sizeof(modulusBits)},
    {CKA_PUBLIC_EXPONENT, publicExponent, sizeof(publicExponent)}
};
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, 5,
    privateKeyTemplate, 8,
  
```

```

    &hPublicKey, &hPrivateKey);
if (rv == CKR_OK) {
    .
    .
}

```

4.12.3. 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
);

```

C_WrapKey «обгортає» (тобто, зашифровує) особистий або секретний ключ. hSession є дескриптором сесії; pMechanism є покажчиком на механізм «обгорнення»/шифрування; hWrappingKey є дескриптором ключа «обгорнення»/шифрування; hKey є дескриптором ключа, що «обгортається»/ зашифровується; pWrappedKey вказує розташування, де отримується зашифрований ключ; і pulWrappedKeyLen є покажчиком на розташування, де утримується довжина зашифрованого ключа.

C_WrapKey використовує прийняті правила (Конвенцію), описані в розділі 4.2.6 по здійсненню виведення даних.

Атрибут CKA_WRAP ключа зашифрування, який вказує чи підтримує ключ операцію «обгорнення», повинен бути встановлений у CK_TRUE. Атрибут CKA_EXTRACTABLE ключа, що «обгортається», повинен бути встановлений у CK_TRUE.

Якщо ключ, який зашифровується, неможливо «обгорнути» з деяких конкретних причин, незважаючи на те, що його атрибут CKA_EXTRACTABLE має значення CK_TRUE, то C_WrapKey не виконується, та видається помилка з кодом CKR_KEY_NOT_WRAPABLE. Якщо неможливо зашифрування за допомогою вказаного ключа зашифрування виключно через його довжину, то C_WrapKey не виконується та формується помилка з кодом CKR_KEY_SIZE_RANGE.

C_WrapKey може використовуватися в таких випадках:

- Для зашифрування будь-якого секретного ключа за допомогою відкритого ключа, що підтримує зашифрування та розшифрування.

- Для зашифрування будь-якого секретного ключа за допомогою будь-якого іншого секретного ключа. Треба враховувати розмір ключа і стійкість механізму, бо токен може не дозволити операцію.
- Зашифрування особистого ключа будь-яким секретним ключем.

Для розподілення ключів «обгорнення» так, що може зашифровуватися тільки підмножина ключів, що вилучаються (extractable), у ключі «обгорнення» може використовуватися шаблон CKA_WRAP_TEMPLATE для визначення множини атрибутів, які буде порівняно з атрибутами ключа для зашифрування. Якщо всі атрибути, відповідно до C_FindObject правил, будуть зіставлені, то буде виконано зашифрування. Значення цих атрибутів є у шаблоні атрибутів, і розмір – це кількість елементів CK_ATTRIBUTE у шаблоні. Якщо цей атрибут відсутній, тоді будь-який шаблон є неприйнятним. Атрибути, які не надані, не перевіряються. Якщо будь-який атрибут є невідповідним при спробі зашифрування ключа, тоді функція повертає CKR_KEY_HANDLE_INVALID.

Повернені	значення:	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.		

Приклад:

```

CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hWrappingKey, hKey;
CK_MECHANISM mechanism = {
    CKM_DES3_ECB, NULL_PTR, 0
};
CK_BYTE wrappedKey[8];
CK_ULONG ulWrappedKeyLen;
CK_RV rv;

.
.
ulWrappedKeyLen = sizeof(wrappedKey);
rv = C_WrapKey(

```

```

    hSession, &mechanism,
    hWrappingKey, hKey,
    wrappedKey, &ulWrappedKeyLen);
if (rv == CKR_OK) {
    .
    .
}

```

4.12.4. 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
);

```

C_UnwrapKey «розгортає» (тобто розшифровує) зашифрований ключ, і створює при цьому новий особистий або секретний ключовий об'єкт.

hSession є дескриптором сесії;
 pMechanism є покажчиком на механізм розгорнення;
 hUnwrappingKey є дескриптором ключа розгорнення;
 pWrappedKey є покажчиком на зашифрований ключ;
 ulWrappedKeyLen – це довжина зашифрованого ключа;
 pTemplate вказує шаблон для нового (розшифрованого) ключа;
 ulAttributeCount – це кількість атрибутів у шаблоні;
 phKey є покажчиком на розташування, де утримується дескриптор розшифрованого ключа.

Атрибут CKA_UNWRAP ключа «розгорнення», який вказує, чи підтримує ключ операцію «розгорнення», повинен бути встановлений у CK_TRUE.

Новий ключ буде мати за замовчуванням атрибут CKA_ALWAYS_SENSITIVE із значенням CK_FALSE і атрибут CKA_NEVER_EXTRACTABLE із значенням CK_FALSE. Атрибут CKA_EXTRACTABLE за замовчуванням встановлений у CK_TRUE.

Деякі механізми можуть змінювати або намагатися змінити вміст структури pMechanism в той же час, коли виконується розшифрування ключа.

Якщо виклик C_UnwrapKey не підтримує визначений шаблон, він не виконується і повертається без створення будь-якого ключового об'єкта.

Ключовий об'єкт, створений при успішному виклику C_UnwrapKey, буде мати атрибут CKA_LOCAL із значенням CK_FALSE.

Для розподілення ключів «розгорнення» так, що вони можуть лише розшифрувати підмножину ключів, атрибут CKA_UNWRAP_TEMPLATE використовується у розшифрованому ключі для визначення набору атрибутів, який буде додано до атрибутів ключа, що розшифровується. Якщо атрибути не суперечать атрибутам вказаного користувачем шаблону у 'pTemplate', то буде виконано розшифрування. Значення цього атрибута є шаблон атрибутів і розмір являє собою кількість елементів CK_ATTRIBUTE у шаблоні. Якщо цей атрибут відсутній у ключі розшифрування, то ніякі додаткові атрибути не додаються. Якщо будь-який атрибут конфліктує з розгорнутим ключем, то функція повертає CKR_TEMPLATE_INCONSISTENT.

Повернені	значення:	
CKR_ATTRIBUTE_READ_ONLY,		CKR_ARGUMENTS_BAD,
CKR_ATTRIBUTE_VALUE_INVALID,		CKR_ATTRIBUTE_TYPE_INVALID,
CKR_CRYPTOKI_NOT_INITIALIZED,		CKR_BUFFER_TOO_SMALL,
CKR_DEVICE_MEMORY,		CKR_DEVICE_ERROR,
CKR_DOMAIN_PARAMS_INVALID,		CKR_DEVICE_REMOVED,
CKR_FUNCTION_FAILED,	CKR_GENERAL_ERROR,	CKR_FUNCTION_CANCELED,
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_UNWRAPPING_KEY_HANDLE_INVALID,		
CKR_UNWRAPPING_KEY_SIZE_RANGE,		
CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT,		
CKR_USER_NOT_LOGGED_IN,		CKR_WRAPPED_KEY_INVALID,
CKR_WRAPPED_KEY_LEN_RANGE,		

Приклад:

```
CK_SESSION_HANDLE hSession;  
CK_OBJECT_HANDLE hUnwrappingKey, hKey;  
CK_MECHANISM mechanism = {  
    CKM_DES3_ECB, NULL_PTR, 0  
};  
CK_BYTE wrappedKey[8] = {...};  
CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;  
CK_KEY_TYPE keyType = CKK_DES;  
CK_BBOOL true = CK_TRUE;  
CK_ATTRIBUTE template[] = {  
    {CKA_CLASS, &keyClass, sizeof(keyClass)},  
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},  
};
```

```

    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_DECRYPT, &true, sizeof(true)}
};
CK_RV rv;

.
.
rv = C_UnwrapKey(
    hSession, &mechanism, hUnwrappingKey,
    wrappedKey, sizeof(wrappedKey), template, 4, &hKey);
if (rv == CKR_OK) {
    .
    .
}

```

4.12.5. 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
);

```

C_DeriveKey виробляє ключ від базового ключа, створюючи новий ключовий об'єкт.

hSession є дескриптор сесії;
pMechanism є покажчик на структуру, яка визначає механізм вироблення ключа;
hBaseKey є дескриптор базового ключа;
pTemplate є покажчик на шаблон для нового ключа;
ulAttributeCount є кількість атрибутів у шаблоні,
phKey є покажчик на місце розташування, яке утримує дескриптор виробленого ключа.

Значення CK_SENSITIVE, CK_ALWAYS_SENSITIVE, CK_EXTRACTABLE і CK_NEVER_EXTRACTABLE атрибутів для базових ключів може впливати на значення цих атрибутів для знову отриманого ключа. Дивіться опис кожного конкретного ключового механізму вироблення для будь-яких обмежень цього типу.

Якщо виклик C_DeriveKey не може підтримувати задані з ним шаблони, то функція не виконується і повинна повернутися без створення будь-яких ключових об'єктів.

Ключовий об'єкт, створений успішним викликом C_DeriveKey матиме CKA_LOCAL атрибут, встановлений в CK_FALSE.

Повертаються значення:

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.

Приклад:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hPublicKey, hPrivateKey, hKey;
CK_MECHANISM keyPairMechanism = {
    CKM_DH_PKCS_KEY_PAIR_GEN, NULL_PTR, 0
};
CK_BYTE prime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE publicKey[128];
CK_BYTE otherPublicValue[128];
CK_MECHANISM mechanism = {
    CKM_DH_PKCS_DERIVE, otherPublicValue, sizeof(otherPublicValue)
};
CK_ATTRIBUTE pTemplate[] = {
    CKA_VALUE, &publicValue, sizeof(publicValue)
};
CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_DES;
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE publicKeyTemplate[] = {
    {CKA_PRIME, prime, sizeof(prime)},
    {CKA_BASE, base, sizeof(base)}
};
CK_ATTRIBUTE privateKeyTemplate[] = {
    {CKA_DERIVE, &true, sizeof(true)}
};
CK_ATTRIBUTE template[] = {
```

```

    {CKA_CLASS, &keyClass, sizeof(keyClass)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_DECRYPT, &true, sizeof(true)}
};
CK_RV rv;

.
.
rv = C_GenerateKeyPair(
    hSession, &keyPairMechanism,
    publicKeyTemplate, 2,
    privateKeyTemplate, 1,
    &hPublicKey, &hPrivateKey);
if (rv == CKR_OK) {
    rv = C_GetAttributeValue(hSession, hPublicKey, &pTemplate, 1);
    if (rv == CKR_OK) {
        /* Put other guy's public value in otherPublicValue */
        .
        .
        rv = C_DeriveKey(
            hSession, &mechanism,
            hPrivateKey, template, 4, &hKey);
        if (rv == CKR_OK) {
            .
            .
        }
    }
}
}

```

4.13. Функції генерування випадкових чисел

Cryptoki надає такі функції для генерації випадкових чисел:

4.13.1. C_SeedRandom

```

CK_DEFINE_FUNCTION(CK_RV, C_SeedRandom)(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pSeed,
    CK_ULONG ulSeedLen
);

```

C_SeedRandom змішує додатковий матеріал зерна (seed) у генераторі випадкових чисел токена.

hSession є дескриптором сесії;

pSeed вказує на матеріал зерна;
ulSeedLen є довжиною в байтах матеріалу зерна.

Повернені	значення:	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_SEED_NOT_SUPPORTED,	
CKR_RANDOM_NO_RNG,		CKR_SESSION_CLOSED,
CKR_SESSION_HANDLE_INVALID,	CKR_USER_NOT_LOGGED_IN.	

Приклад: див C_GenerateRandom.

4.13.2. C_GenerateRandom

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

C_GenerateRandom генерує випадкові або псевдовипадкові дані.
hSession є дескриптором сесії;
pRandomData є покажчик на місце розташування, де утримуються випадкові дані;
ulRandomLen є довжиною в байтах випадкових або псевдовипадкових даних, які генеруються.

Повернені	значення:	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.		

Приклад:

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

```

.
.
rv = C_SeedRandom(hSession, seed, sizeof(seed));
if (rv != CKR_OK) {
.
.
}
rv = C_GenerateRandom(hSession, randomData, sizeof(randomData));
if (rv == CKR_OK) {
.
.
}

```

5. Механізми

5.1. Загальні визначення

Механізм (Mechanism) вказує те, як саме повинен бути виконаний певний криптографічний процес/ операція.

Ці Технічні Специфікації визначають вимоги до реалізації засобів криптографічного захисту інформації, що підтримують національні крипто алгоритми ДСТУ ГОСТ 28147:2009, ГОСТ 34.311-95, ДСТУ 4145-2002 відповідно до стандартів ДСТУ ГОСТ 28147:2009, ГОСТ 34.311-95, ДСТУ 4145-2002 та міжнародного стандарту PKCS#11. Вимоги до реалізації інших крипто алгоритмів повинні відповідати зазначеним у стандарті PKCS#11 версії 2.20.

Таблиця 5.1 показує, які механізми підтримують які криптографічні операції. Будь-який конкретний токен може підтримувати усі чи тільки підмножину механізмів у наведеному списку. Механізми, які підтримує конкретний токен, повинні бути визначені у його експлуатаційній документації.

Таблиця 5.1. Механізми проти функцій

Механізм	Функції					
	Encrypt & Decrypt	Sign & Verify	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_GOST28147_KEY_GEN				✓		
CKM_GOST28147_ECB	✓					
CKM_GOST28147_OFB	✓					
CKM_GOST28147_CFB	✓					
CKM_GOST28147_MAC		✓				
CKM_GOST28147_WRAP					✓	
CKM_GOST28147_UNWRAP					✓	

Механізм	Функції					
	Encrypt & Decrypt	Sign & Verify	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_GOST34311			✓			
CKM_DSTU4145_KEY_PAIR_GEN				✓		
CKM_DSTU4145		✓				
CKM_DSTU4145_WITH_GOST34311		✓				
CKM_DSTU4145_ECDH_DERIVE						✓
CKM_DSTU4145_ECDH_COFACTO R_DERIVE						✓

Слід підкреслити, що значення атрибутів механізмів повинні бути у форматах, визначених у розділі 3.1, зокрема, значення байт масивів (тип CK_BYTE) повинно бути представлено у форматі Big-Endian (MSB - most significant byte first, від старшого до молодшого, старший байт зберігається першим).

5.2. Об'єкт SBOX

Об'єкт SBOX зберігає параметр алгоритму ГОСТ 28147 - значення таблиці заповнення вузлів заміни блоків підстановки довгострокових ключових елементів (ДКЕ) у атрибуті CKA_VALUE об'єкту класу даних SKO_DATA.

Цей параметр алгоритму ГОСТ 28147 використовуються як атрибут ключів рядом механізмів, і може міститися у окремому об'єкті даних. Ідентифікатор атрибуту ключів визначається таким чином:

CKA_SBOX = CKA_VENDOR_DEFINED + 0x00420111;

де

CKA_VENDOR_DEFINED = 0x80000000.

Значенням атрибута CKA_SBOX є DER-кодована ASN.1 структура:

```
SBOX ::= CHOICE {
    namedsbox OBJECT IDENTIFIER, -- об'єктний ідентифікатор ДКЕ
    sbboxbinary OCTET STRING, -- стисле зображення елементу ДКЕ (64 bytes)
}
```

Об'єкт SBOX повинен підтримувати такі атрибути:

Таблиця 5.2. Атрибути, що підтримуються об'єктом SBOX

Атрибути	Тип даних	Значення
CKA_CLASS ¹	CK_OBJECT_CLASS	Клас об'єкта
CKA_VALUE ^{1,7}	Byte array	Значення ДКЕ - стисле зображення, завжди 64 байта (не пусте значення)

Атрибути	Тип даних	Значення
CKA_VALUE_LEN ²	CK_ULONG	Довжина значення атрибута CKA_VALUE в байтах (заповнюється автоматично при створенні об'єкту)
CKA_OBJECT_ID ¹	Byte array	DER-кодоване значення ASN.1 структури об'єктного ідентифікатора ДКЕ (не пусте значення)
CKA_LABEL	RFC2279 рядок	Опис об'єкту (за замовчуванням пусто).
CKA_TOKEN	CK_BBOOL	CK_TRUE, якщо об'єкт є об'єктом токена; CK_FALSE якщо об'єкт є сесійним.
CKA_PRIVATE	CK_BBOOL	CK_TRUE, якщо об'єкт є особистим об'єктом; CK_FALSE, якщо об'єкт є відкритим об'єктом.
CKA_SENSITIVE ^{1,11}	CK_BBOOL	CK_TRUE, якщо атрибут об'єкту CKA_VALUE є чутливим (захищений від перегляду)
CKA_EXTRACTABLE ^{1,12}	CK_BBOOL	CK_TRUE, якщо об'єкт є таким, що вилучається, та може бути «обгорнутим»/ зашифрованим
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE, якщо об'єкт може бути змінений

Примітки див. у Таблиці 3.1.

Також:

- Якщо CKA_PRIVATE = CK_FALSE, то об'єкт є видимим для неавторизованого користувача.
- Якщо CKA_SENSITIVE = CK_FALSE, то значення ДКЕ (CKA_VALUE) є видимим (може бути отримано функцією C_GetAttributeValue).
- Якщо CKA_EXTRACTABLE = CK_FALSE, то значення ДКЕ (CKA_VALUE) не може бути обгорнено функцією C_WrapKey.
- Якщо CKA_MODIFIABLE = CK_FALSE, то об'єкт не може бути модифікований (змінений) після його створення.

При використанні цього об'єкту у якості атрибуту інших об'єктів (ключі) чи механізмами, їх відповідний параметр повинен отримати значення об'єктного ідентифікатора (namedsbox) у атрибуті CKA_SBOX, що співпадає із значенням атрибута CKA_OBJECT_ID об'єкту SBOX.

Для об'єктів SBOX не передбачено механізму генерації.

У таблиці 5.3 наведено обов'язкові атрибути шаблону об'єкта SBOX для створення об'єкта функцією C_CreateObject.

Таблиця 5.3. Обов'язкові атрибути шаблону об'єкта SBOX (значення наведені у якості зразку і не є обов'язковими)

Атрибути	Значення
СКА_CLASS	СКО_DATA
СКА_VALUE	Значення ДКЕ (64 байта)
СКА_OBJECT_ID	Об'єктний ідентифікатор значення ДКЕ
СКА_SENSITIVE	CK_TRUE
СКА_EXTRACTABLE	CK_FALSE

Шаблон може містити додаткові атрибути, які дозволяються для об'єктів класу СКО_DATA.

Для створення секретного ДКЕ необхідно встановлювати СКА_PRIVATE = CK_TRUE, СКА_SENSITIVE = CK_TRUE. У цьому випадку ДКЕ як об'єкт є невидимим для неавторизованого користувача, а його значення є завжди невидимим.

При створенні об'єкту SBOX, якщо заданий об'єктний ідентифікатор (СКА_OBJECT_ID) вже існує на токени, повертається помилка CKR_FUNCTION_CANCELED.

При видаленні об'єкту SBOX з токени, якщо поточний об'єктний ідентифікатор (СКА_OBJECT_ID) задіяно в будь-якому ключовому об'єкті в атрибуті (СКА_SBOX) на токени, повертається помилка CKR_FUNCTION_CANCELED.

Для об'єктів SBOX повинні застосовуватися, щонайменше, такі функції: C_CreateObject, C_DestroyObject, C_GetAttributeValue, C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal.

Приклад шаблону для створення об'єкту:

```
CK_BYTE SBox[64] = {
    0xa9,0xd6,0xeb,0x45,0xf1,0x3c,0x70,0x82,
    0x80,0xc4,0x96,0x7b,0x23,0x1f,0x5e,0xad,
    0xf6,0x58,0xeb,0xa4,0xc0,0x37,0x29,0x1d,
    0x38,0xd9,0x6b,0xf0,0x25,0xca,0x4e,0x17,
    0xf8,0xe9,0x72,0x0d,0xc6,0x15,0xb4,0x3a,
    0x28,0x97,0x5f,0x0b,0xc1,0xde,0xa3,0x64,
    0x38,0xb5,0x64,0xea,0x2c,0x17,0x9f,0xd0,
    0x12,0x3e,0x6d,0xb8,0xfa,0xc5,0x79,0x04
};
```

```
CK_BYTE OID[] = {0x06,0x0c,0x2a,0x86,0x24,0x02,0x01,
    0x01,0x01,0x01,0x01,0x01,0x02,0x10};
```

```

CK_BYTE          LabelSbox[]    = "SBOX";
CK_OBJECT_CLASS  ClassSbox      = CKO_DATA;
CK_BBOOL         Token          = True;
CK_ATTRIBUTE     TemplateSbox[] = {
    {CKA_LABEL,          LabelSbox,      sizeof(LabelSbox)-1},
    {CKA_CLASS,          &ClassSbox,     sizeof(ClassSbox)},
    {CKA_TOKEN,          &Token,         sizeof(CK_BBOOL)},
    {CKA_PRIVATE,        &False,         sizeof(CK_BBOOL)},
    {CKA_VALUE,          SBox,           sizeof(SBox)},
    {CKA_SENSITIVE,      &False,         sizeof(CK_BBOOL)},
    {CKA_EXTRACTABLE,    &False,         sizeof(CK_BBOOL)},
    {CKA_OBJECT_ID,      OID,            sizeof(OID)}
};

```

5.3. ДСТУ ГОСТ 28147:2009

5.3.1. Визначення

Цей розділ визначає ключі типу СКК_GOST28147 та такі механізми:

СКМ_GOST28147_KEY_GEN – генерація ключів;

СКМ_GOST28147_ECB - шифрування у режимі простої заміни (ECB);

СКМ_GOST28147_OFB - шифрування у режимі гамування (OFB);

СКМ_GOST28147_CFB - шифрування у режимі гамування із зворотнім зв'язком (CFB);

СКМ_GOST28147_MAC – обчислення/перевіряння імітовставки (MAC);

СКМ_GOST28147_WRAP – шифрування ключів (WRAP).

5.3.2. Ключові об'єкти

Ключові об'єкти ДСТУ ГОСТ 28147:2009 є об'єктами класу СКА_CLASS, що повинен мати значення «СКО_SECRET_KEY», та типом СКА_KEY_TYPE, що повинен мати значення СКК_GOST28147, ідентифікатор якого визначається таким чином:

СКК_GOST28147 = СКК_VENDOR_DEFINED + 0x00420111,

де

СКК_VENDOR_DEFINED = 0x80000000.

Окрім атрибутів СКА_CLASS і СКА_KEY_TYPE, та стандартних атрибутів, визначених для об'єктів «Секретні ключі» (див. розділ 3.7.9), ці ключі повинні підтримувати такі атрибути:

Таблиця 5.4. Атрибути, що підтримуються ключами ДСТУ ГОСТ 28147:2009

Атрибути	Тип даних	Значення
CKA_CLASS	CK_OBJECT_CLASS	Клас ключа
CKA_KEY_TYPE	CK_KEY_TYPE	Тип ключа
CKA_SBOX ¹	Byte array	DER-кодоване значення ASN.1 структури об'єкта SBOX (див. розділ 5.2)
CKA_VALUE ¹	Byte array	Значення ключа (32 байта)
CKA_VALUE_LEN ²	CK_ULONG	Довжина значення атрибута CKA_VALUE в байтах (заповнюється автоматично при створенні об'єкту)
CKA_LABEL	RFC2279 рядок	Опис об'єкту (за замовчуванням пусто).
CKA_ID ⁹	Byte array	Унікальний ідентифікатор ключа (схема виробництва визначається виробником)
CKA_ENCRYPT ¹	CK_BBOOL	CK_TRUE, якщо ключ підтримує зашифрування
CKA_DECRYPT ¹	CK_BBOOL	CK_TRUE, якщо ключ підтримує розшифрування
CKA_SIGN	CK_BBOOL	CK_TRUE, якщо ключ підтримує підписування, де підпис є додатком до даних
CKA_VERIFY	CK_BBOOL	CK_TRUE, якщо ключ підтримує верифікацію (перевіряння) підпису, де підпис додається до даних
CKA_WRAP	CK_BBOOL	CK_TRUE, якщо ключ підтримує «обгорнен-ня», тобто може бути використаний для зашифрування іншого ключа
CKA_UNWRAP	CK_BBOOL	CK_TRUE, якщо ключ підтримує «розгорнен-ня», тобто може бути використаний для розшифрування іншого ключа
CKA_TOKEN	CK_BBOOL	CK_TRUE, якщо об'єкт є об'єктом токена; CK_FALSE якщо об'єкт є сесійним.
CKA_PRIVATE	CK_BBOOL	CK_TRUE, якщо об'єкт є

Атрибути	Тип даних	Значення
		особистим об'єктом; CK_FALSE, якщо об'єкт є відкритим об'єктом.
CKA_SENSITIVE ^{1,11}	CK_BBOOL	CK_TRUE, якщо атрибут об'єкту CKA_VALUE є чутливий (захищений від перегляду)
CKA_EXTRACTABLE ^{1,12}	CK_BBOOL	CK_TRUE, якщо ключ є таким, що вилучається, та може бути «обгорнутим»/зашифрованим
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE, якщо об'єкт може бути змінений
CKA_LOCAL ^{2,4,6}	CK_BBOOL	CK_TRUE, тільки якщо ключ був або згенерований локально (тобто, у токени) шляхом виклику C_GenerateKey або C_GenerateKeyPair, або створено шляхом виклику C_CopyObject, як копія ключа, який мав атрибут CKA_LOCAL, що встановлено у CK_TRUE

Примітки див. у Таблиці 3.1.

Атрибути за замовчуванням створюють ключ, призначений для шифрування, обчислення MAC, але не призначений для WRAP захисту ключа.

Якщо атрибут CKA_SBOX не задано, то механізмами повинен використовуватися ДКЕ №1 Наказу №114, як атрибут за замовчуванням.

У разі, якщо атрибут CKA_SBOX зберігає «невідоме» значення об'єктного ідентифікатора (тобто, що офіційно не опубліковано), то автоматично виконується пошук об'єкта SBOX на токени з відповідним об'єктним ідентифікатором у його атрибуті CKA_OBJECT_ID.

Для створення ключа ДСТУ ГОСТ 28147:2009 передбачено механізм генерації СКМ_GOST28147_KEY_GEN. Також створення ключа ДСТУ ГОСТ 28147:2009 можливе за допомогою функції C_CreateObject. У таблиці 5.5 наведено обов'язкові атрибути шаблону для створення ключа.

Таблиця 5.5. Обов'язкові атрибути шаблону ключа ДСТУ ГОСТ 28147:2009 (значення наведені у якості зразку і не є обов'язковими)

Атрибути	Значення
----------	----------

Атрибути	Значення
CKA_CLASS	CKO_SECRET_KEY
CKA_KEY_TYPE	CKK_GOST28147
CKA_SBOX	OID_GOST28147_SBOX_1 (ДКЕ №1 Наказу №114)
CKA_VALUE	Значення ключа (32 байта)
CKA_ID	Унікальний ідентифікатор ключа
CKA_SENSITIVE	CK_TRUE
CKA_EXTRACTABLE	CK_FALSE

До ключових об'єктів CKK_GOST28147 повинні застосовуватися, щонайменше, такі функції: C_CreateObject, C_DestroyObject, C_GetAttributeValue, C_SetAttributeValue, C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal.

Приклад шаблону для створення об'єкту:

```

CK_BYTE ID[] = {0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x10,
0x11,0x12,0x13,0x14};
CK_BYTE sbox_OID[] = {0x06,0x0C,0x2A,0x86,0x24,0x02,0x01,0x01,0x01,
0x01,0x01,0x01,0x02,0x10};
CK_BYTE val[] = {0x77,0xA7,0xDC,0x87,0x72,0x43,0x3C,0x60,0x14,0x8F,
0xC8,0x65,0x26,0x60,0xC3,0x97,0xDC,0x2F,0xA6,0x8A,0x7B,0x3E,0x73,0x7A,
0xE9,0xC7,0x0D,0xED,0xAD,0xF4,0xE0,0x0A };

CK_KEY_TYPE      KeyTypeSecret    = CKK_GOST28147;
CK_OBJECT_CLASS  ClassSecretKey   = CKO_SECRET_KEY;
CK_BYTE          LabelStr[]        = "Test Create Gost28147 Key";

CK_ATTRIBUTE      TemplGostKey[] = {
    {CKA_LABEL,      LabelStr,      sizeof(LabelStr)-1},
    {CKA_KEY_TYPE,   &KeyTypeSecret, sizeof(CK_KEY_TYPE)},
    {CKA_CLASS,      &ClassSecretKey, sizeof(CK_OBJECT_CLASS)},
    {CKA_SBOX,        sbox_OID,      sizeof(sbox_OID)},
    {CKA_VALUE,       val,           sizeof(val)},
    {CKA_TOKEN,       &True,         sizeof(CK_BBOOL)},
    {CKA_PRIVATE,     &False,        sizeof(CK_BBOOL)},
    {CKA_ENCRYPT,      &True,         sizeof(CK_BBOOL)},
    {CKA_DECRYPT,      &True,         sizeof(CK_BBOOL)},
    {CKA_EXPORTABLE, &True,         sizeof(CK_BBOOL)},
    {CKA_EXTRACTABLE, &True,         sizeof(CK_BBOOL)},
    {CKA_SENSITIVE,   &False,        sizeof(CK_BBOOL)},
    {CKA_ID,          ID,            sizeof(ID)}
};

```

5.3.3. Генерація ключів

Механізм генерації ключів типу CKM_GOST28147, що позначається CKM_GOST28147_KEY_GEN, є механізмом генерації, що відповідає стандарту ДСТУ ГОСТ 28147:2009.

Ідентифікатор механізму:

CKM_GOST28147_KEY_GEN = CKM_VENDOR_DEFINED + 0x00420041;

де

CKM_VENDOR_DEFINED = 0x80000000;

Інформація про механізм CKM_GOST28147_KEY_GEN, що повертається структурою CK_MECHANISM_INFO, повинна бути такою:

ulMinKeySize = 256;

ulMaxKeySize = 256;

Flags = CKF_GENERATE [| CKF_HW];

Примітки:

1. Довжина ключів визначається в бітах.
2. Додатковий прапор [CKF_HW] повинен встановлюватися у разі апаратної реалізації.

Механізм при відсутності деяких (або всіх) атрибутів у шаблоні функції генерування ключа C_GenerateKey встановлює за замовчуванням атрибути нового ключа, визначені у Таблиці 5.6:

Таблиця 5.6. Атрибути за замовчуванням механізму генерації ключів CKM_GOST28147_KEY_GEN

Атрибути	Значення
CKA_CLASS	CKO_SECRET_KEY
CKA_KEY_TYPE	CKK_GOST28147
CKA_SBOX ¹	OID_GOST28147_SBOX_1 (ДКЕ №1 Наказу №114)
CKA_VALUE ¹	Сгенероване значення ключа (32 байта)
CKA_VALUE_LEN ²	32
CKA_LABEL	«Gost 28147 Secret Key»
CKA_ID ⁹	Унікальний ідентифікатор ключа
CKA_ENCRYPT ¹	CK_TRUE
CKA_DECRYPT ¹	CK_TRUE
CKA_SIGN	CK_TRUE
CKA_VERIFY	CK_TRUE
CKA_WRAP	CK_FALSE
CKA_UNWRAP	CK_FALSE

Атрибути	Значення
CKA_TOKEN	CK_FALSE
CKA_PRIVATE	CK_TRUE
CKA_SENSITIVE ^{1,11}	CK_TRUE
CKA_EXTRACTABLE ^{1,12}	CK_FALSE
CKA_MODIFIABLE	CK_TRUE

Примітки див. у Таблиці 3.1.

Шаблон за замовчуванням призначено для ключа, що може виконувати функції зашифрування та розшифрування, обчислення та перевіряння MAC (імітовставки), ключ є сесійним, чутливим та не може вилучатися. Ключ не може застосовуватися для функцій зашифрування (WRAP) та розшифрування (UNWRAP) ключів.

Механізм може мати чи не мати параметрів (див. типи даних для механізму, розділ 2.7.5).

Необовязкові параметри механізму генерації, а саме: початкове значення генератора випадкових чисел, - задаються такою структурою:

```
typedef struct CK_SEED_PARAMS {
    CK_BYTE      seed[64];
} CK_SEED_PARAMS;
typedef CK_SEED_PARAMS * CK_SEED_PARAMS_PTR;
```

Якщо застосовується механізм без параметрів (NULL), то початкове значення генератора випадкових чисел може бути встановлено окремим викликом функції C_SeedRandom, що передує механізму генерації.

Приклад генерації ключа:

```
CK_SESSION_HANDLE hSession = ...;    // handle of session
```

```
CK_BYTE sbox_OID[] = {0x06,0x0C,0x2A,0x86,0x24,0x02,0x01,0x01,
    0x01,0x01,0x01,0x01,0x02,0x10};
```

```
CK_KEY_TYPE keyTypeSecret = CKK_GOST28147;
CK_OBJECT_CLASS classSecretKey = CKO_SECRET_KEY;
CK_BYTE labelStr[] = "Test Generate Key";
```

```
CK_ATTRIBUTE templGostKey[] = {
    {CKA_KEY_TYPE, &keyTypeSecret, sizeof(CK_KEY_TYPE)},
    {CKA_CLASS, &classSecretKey, sizeof(CK_OBJECT_CLASS)},
    {CKA_SENSITIVE, &False, sizeof(CK_BBOOL)},
    {CKA_TOKEN, &True, sizeof(CK_BBOOL)},
    {CKA_EXPORTABLE, &True, sizeof(CK_BBOOL)},
```

```

    {CKA_EXTRACTABLE, &True, sizeof(CK_BBOOL)},
    {CKA_SBOX, sbox_OID, sizeof(sbox_OID)},
    {CKA_LABEL, labelStr, sizeof(labelStr)-1}}
};

CK_BYTE seed[64] = {0x77,0xA7,0xDC,0x87,0x72,0x43,0x3C,0x60,
    0x14,0x8F,0xC8,0x65,0x26,0x60,0xC3,0x97,0xDC,0x2F,
    0xA6,0x8A,0x7B,0x3E,0x73,0x7A,0xE9,0xC7,0x0D,0xED,
    0xAD,0xF4,0xE0,0x0A,0x77,0xA7,0xDC,0x87,0x72,0x43,
    0x3C,0x60,0x14,0x8F,0xC8,0x65,0x26,0x60,0xC3,0x97,
    0xDC,0x2F,0xA6,0x8A,0x7B,0x3E,0x73,0x7A,0xE9,0xC7,
    0x0D,0xED,0xAD,0xF4,0xE0,0x0A
};

CK_MECHANISM mech ;
mech.mechanism = CKM_GOST28147_KEY_GEN;
mech.pParameter = seed;
mech.parameterLen = sizeof(seed);

CK_OBJECT_HANDLE hSecretKey = NULL;

rv = C_GenerateKey(hSession, &mech, templGostKey,
    NUMITEMS(templGostKey), &hSecretKey);

if (rv)
    printf("\nC_GenerateKey failed: %d", rv);
else
    printf("\nGostKey was saved on token. handle - %d", hSecretKey);

```

5.3.4. Механізми шифрування

Пристрої, що підтримують шифрування за ДСТУ ГОСТ 28147:2009, повинні повертати у списку механізмів (функція «C_GetMechanismList()») такі ідентифікатори механізмів:

```

CKM_GOST28147_ECB      = CKM_VENDOR_DEFINED + 0x00420011;
CKM_GOST28147_OFB      = CKM_VENDOR_DEFINED + 0x00420012;
CKM_GOST28147_CFB      = CKM_VENDOR_DEFINED + 0x00420013;
де
CKM_VENDOR_DEFINED = 0x80000000.

```

Механізми шифрування повинні підтримувати функції зашифрування та розшифрування як у одно-, так і мульти-частинних операціях (див розділи 4.7, 4.8).

Інформація про механізми, що повертається структурою CK_MECHANISM_INFO, повинна бути такою:

```
ulMinKeySize    = 256;  
ulMaxKeySize    = 256;  
Flags           = CKF_ENCRYPT | CKF_DECRYPT [| CKF_HW ];
```

Примітки:

1. Довжина ключів визначається в бітах.
2. Додатковий прапор [CKF_HW] повинен встановлюватися у разі апаратної реалізації.

Параметри механізмів задаються такою структурою:

```
typedef struct CK_GOST28147_PARAMS {  
    CK_BYTE    iv[8];  
} CK_GOST28147_PARAMS;
```

```
typedef CK_GOST28147_PARAMS* CK_GOST28147_PARAMS_PTR;
```

де поля мають такі значення:

iv– синхропосилка;

Примітки:

- 1) Механізм CKM_GOST28147_ECB ігнорує параметри механізму.
- 2) Довгостроковий ключовий елемент механізму SBOX задається відповідним атрибутом ключа (СКА_SBOX, див. розділ 5.2).
- 3) Параметри за замовчуванням:
 - синхропосилка, що заповнена нулями.

Параметри за замовчуванням застосовуються за відсутності явно вказаних параметрів.

Ключ типу CKK_GOST28147 повинен для функції зашифрування мати атрибут СКА_ENCRYPT, встановлений у CK_TRUE, та для функції розшифрування атрибут СКА_DECRYPT, встановлений у CK_TRUE.

Обмеження на розмір вхідних буферів залежить від реалізації криптографічного пристрою. У разі нехватки пам'яті на пристрої для приймання вхідних даних, буде повернено помилку CKR_DATA_LEN_RANGE.

Приклад операції зашифрування за механізмом CKM_GOST28147_CFB:

```
CK_SESSION_HANDLE hSession = ...; // handle of session
```

```
CK_SIZE lenText;
```

```

CK_BYTE text[] = {0x55,0x55,0x55,0x55,0xAA,0xAA,0xAA,0xAA};
CK_BYTE iv[8] = {0x2A,0x80,0xA7,0xC3,0xFF,0xA8,0xE3,0x47};

CK_GOST28147_PARAMS_PTR pParamGost28147 =
    (CK_GOST28147_PARAMS_PTR)
        calloc(sizeof(CK_GOST28147_PARAMS),1);
memcpy(pParamGost28147->iv, iv, sizeof(iv));

CK_MECHANISM_PTR pMechGost28147 =
    (CK_MECHANISM_PTR)
        calloc(sizeof(CK_MECHANISM),1);
pMechGost28147->pParameter = pParamGost28147;
pMechGost28147->parameterLen = sizeof(CK_GOST28147_PARAMS);
pMechGost28147->mechanism = CKM_GOST28147_CFB;

rv = C_EncryptInit(hSession, pMechGost28147, hSecretKey);
if (rv) printf("\nC_EncryptInit failed: %d", rv);

lenText = sizeof(text);

rv = C_Encrypt(hSession, text, lenText, text, &lenText);
if (rv) printf("\nC_Encrypt failed: %d", rv);

```

5.3.5. Механізм імітовставки

Механізм вироблення імітовставки (MAC), відповідно до стандарту ДСТУ ГОСТ 28147:2009, задається таким ідентифікатором:

```

CKM_GOST28147_MAC      = CKM_VENDOR_DEFINED + 0x00420014,
де
CKM_VENDOR_DEFINED = 0x80000000.

```

Інформація про механізм, що повертається структурою CK_MECHANISM_INFO, повинна бути такою:

```

ulMinKeySize      = 256;
ulMaxKeySize      = 256;
Flags              = CKF_SIGN | CKF_VERIFY[| CKF_HW ];

```

Примітки:

1. Довжина ключів визначається в бітах.
2. Додатковий прапор [CKF_HW] повинен встановлюватися у разі апаратної реалізації.

Механізм повинен підтримувати функції підписування (формування MAC) та перевіряння підпису (MAC) у режимах як одно-, так і мульти-частинних операцій (див розділи 4.10, 4.11).

Механізм не має параметрів (NULL).

Ключ типу CKK_GOST28147 повинен для функції формування MAC мати атрибут CKA_SIGN, встановлений у CK_TRUE, та для функції перевіряння MAC атрибут CKA_VERIFY, встановлений у CK_TRUE.

Обмеження на розмір вхідних буферів залежить від реалізації криптографічного пристрою. У разі нехватки пам'яті на пристрої для приймання вхідних даних, буде повернено помилку CKR_DATA_LEN_RANGE.

Приклад обчислення та перевіряння імітовставки за механізмом CKM_GOST28147_MAC:

```
CK_SESSION_HANDLE hSession = ...; // handle of session
CK_OBJECT_HANDLE hSecretKey = ...; // handle of key

CK_ULONG InputText[4] = {0x55555555, 0xAAAAAAAA,
                          0x55555555, 0xCCCCCCCC};

CK_ULONG lenMAC, lenTemp;
CK_BYTE MAC[32];
CK_MECHANISM_PTR pMechGost28147;
CK_GOST28147_PARAMS_PTR pParamGost28147;

pParamGost28147 = (CK_GOST28147_PARAMS_PTR)
    malloc(sizeof(CK_GOST28147_PARAMS));
if (pParamGost28147 == NULL) printf("\nInsufficient memory available\n");

memset(pParamGost28147->iv8, 0, 8);

pMechGost28147 = (CK_MECHANISM_PTR)
    calloc(sizeof(CK_MECHANISM), 1);
if (pMechGost28147 == NULL) printf("\nInsufficient memory available\n");

pMechGost28147->pParameter = pParamGost28147;
pMechGost28147->parameterLen = sizeof(CK_GOST28147_PARAMS);

pMechGost28147->mechanism = CKM_GOST28147_MAC;

rv = C_SignInit(hSession, pMechGost28147, hSecretKey);
if (rv != CKR_OK) printf("\nC_SignInit failed: %d", rv);
```

```

lenMAC = GOST28147_MAC_LENGTH;
rv = C_Sign(hSession, (CK_BYTE_PTR)InputText, sizeof(InputText),
            MAC, &lenMAC);
if (rv != CKR_OK) printf("\nC_Sign failed: %d", rv);

rv = C_VerifyInit(hSession, pMechGost28147, hSecretKey);
if (rv != CKR_OK) printf("\nC_VerifyInit failed: %d", rv);

rv = C_Verify(hSession, (CK_BYTE_PTR)InputText, sizeof(InputText),
              MAC, sizeof(MAC));
if (rv != CKR_OK) printf("\nC_Verify failed: %d", rv);
else printf("\nMAC Verified and OK!");

```

5.3.6. Механізм WRAP

5.3.6.1. Визначення механізму

WRAP механізм, що позначається як CKM_GOST28147_WRAP, використовується для «обгорнення» (зашифрування, WRAP) та «розгорнення» (розшифрування, UNWRAP) ключів (див. розділи 4.12.3, 4.12.4).

Пристрої, що підтримують механізм, повинні повертати у списку механізмів (функція «C_GetMechanismList()») такий ідентифікатор механізму:

```

CKM_GOST28147_WRAP = CKM_VENDOR_DEFINED + 0x00420015;
де
CKM_VENDOR_DEFINED = 0x80000000.

```

Інформація про механізм, що повертається структурою CK_MECHANISM_INFO, повинна бути такою:

```

ulMinKeySize    = 256;
ulMaxKeySize    = 256;
Flags           = CKF_WRAP | CKF_UNWRAP [| CKF_HW ];

```

Примітки:

1. Довжина ключів визначається в бітах.
2. Додатковий прапор [CKF_HW] повинен встановлюватися у разі апаратної реалізації.

Ключ шифрування ключа (КШК), який передається у функцію WRAP/UNWRAP, повинен задовольняти таким вимогам:

```

атрибут CKA_CLASS = CKO_SECRET_KEY;
атрибут CKA_KEY_TYPE = CKK_GOST28147.

```

Ключ КШК типу CKK_GOST28147 повинен для функції «обгорнення»/ WRAP («C_WrapKey()») мати атрибут SKA_WRAP, встановлений у SK_TRUE, та для функції «розгорнення»/ UNWRAP («C_UnwrapKey()») мати атрибут SKA_UNWRAP, встановлений у SK_TRUE. Ключ, який підлягає зашифруванню, повинен мати атрибут SKA_EXTRACTABLE = SK_TRUE (див. розділи 4.12.3, 4.12.4).

Параметри механізму можуть задаватися структурою SK_GOST28147_PARAMS (див. розділ 5.3.4) чи бути відсутні (NULL).

У шаблоні атрибутів, який буде вміщувати розшифрований ключ, при відсутності явно заданих атрибутів будуть прийняті такі атрибути за замовчуванням:

Таблиця 5.7. Набір атрибутів за замовчуванням механізму розгорнення ключів SKM_GOST28147_WRAP, які створюються функцією C_UnwrapKey().

Атрибут	Значення
SKA_KEY_TYPE	CKK_GOST28147
SKA_CLASS	CKO_SECRET_KEY
SKA_SBOX	OID_GOST28147_SBOX_1 (ДКЕ №1 Наказу №114)
SKA_LABEL	"Gost 28147 unwrapped key"
SKA_VALUE_LEN	32
SKA_ENCRYPT	SK_TRUE
SKA_DECRYPT	SK_TRUE
SKA_SIGN	SK_TRUE
SKA_VERIFY	SK_TRUE
SKA_WRAP	SK_FALSE
SKA_UNWRAP	SK_FALSE
SKA_TOKEN	SK_FALSE
SKA_PRIVATE	SK_TRUE
SKA_SENSITIVE ¹¹	SK_TRUE
SKA_EXTRACTABLE ¹²	SK_FALSE
SKA_MODIFIABLE	SK_TRUE

Примітки див. у Таблиці 3.1.

А також:

- Довжина створеного функцією C_UnwrapKey() ключа SKA_VALUE_LEN за замовчуванням дорівнює 32 байти. Якщо функцією C_UnwrapKey() створюється ключ іншої довжини (наприклад, AES 128, 192 біт), то цей атрибут необхідно явно задавати.

Алгоритм GOST28147Wrap механізму SKM_GOST28147_WRAP ґрунтується на стандарті ДСТУ ГОСТ 28147:2009 і міжнародних рекомендаціях RFC 3217.

Алгоритм GOST28147Wrap призначений для шифрування ключових даних чи інших даних, що підлягають захисту (“ключові дані”), використовуючи стандарт ДСТУ ГОСТ 28147:2009 в режимі CFB (гамування із зворотнім зв’язком, відповідно до розділу 4 ДСТУ ГОСТ 28147:2009, надалі - GOST28147-CFB).

Алгоритм GOST28147Wrap призначений також для забезпечення цілісності зашифрованих ключових даних.

5.3.6.2. Процес зашифрування (WRAP)

Процес зашифрування є реалізацією функції C_WrapKey цього механізму.

Вхідними даними процесу зашифрування є:

“dke” – довготривалий ключовий елемент (ДКЕ);

“КЕК” – ключ шифрування ключа (КШК);

“СЕК” – ключові дані для зашифрування (у операції формування “захищені дані” – це ключ шифрування даних, КШД).

Вихідними даними процесу зашифрування є:

“result” – зашифровані ключові дані.

Процес зашифрування виконується у такі кроки:

а) виконати ініціалізацію алгоритму вхідними даними “КЕК”. Шаблон атрибутів ключа “КЕК” повинен містити як атрибут ДКЕ (SBOX об’єкт);

б) обчислити контрольну суму ключових даних “СЕК”. Контрольна сума ключових даних (позначена як “ICV”) призначена для контролю правильності розшифрування зашифрованих ключових даних, та обчислюється як імітовставка довжини 32 біта (“MAC32”) згідно з розділом 5 ДСТУ ГОСТ 28147:2009.

$ICV = MAC32(СЕК, dke, КЕК)$ [4 байти].

Ймовірність “нав’язування невірних даних” (помилки при оцінюванні цілісності ключових даних) відповідно до ДСТУ ГОСТ 28147:2009 становить $2^{(-32)} = 0,0000000002$;

в) виконати конкатенацію ключових даних з отриманою контрольною сумою:

$СЕКICV = СЕК \parallel ICV$;

г) згенерувати випадкові 8 байт, як вектор ініціалізації (синхропосилка) (позначено як “IV”);

д) виконати зашифрування даних “СЕКICV” алгоритмом ДСТУ ГОСТ 28147:2009 в режимі гамування із зворотнім зв’язком (GOST28147-CFB), використовуючи “КЕК”, що встановлені на кроці 1, та вектор ініціалізації “IV”, отриманий за результатами виконання пункту 6.8.2.2) г) Технічних специфікацій:

$TEMP1 = GOST28147-CFB_encrypt(СЕКICV, IV, dke, КЕК)$;

Довжина вихідних даних “TEMP1” дорівнює довжині “CEKICV”;

є) виконати конкатенацію:

TEMP2 = IV || TEMP1;

ж) виконати реверсне перетворення порядку байтів TEMP2, так що перший байт TEMP2 стає останнім байтом, і т.д. Результат перетворення позначимо TEMP3;

з) зашифрувати TEMP3 алгоритмом ДСТУ ГОСТ 28147:2009 в режимі гамування із зворотнім зв'язком (GOST28147-CFB), використовуючи “dke” та ключ “КЕК” та вектор ініціалізації “IV1”:

IV1 = 0x4adda22c79e82105.

Результатом зашифрування алгоритмом GOST28147Wrap є:

result = GOST28147-CFB_encrypt(TEMP3, IV1, dke, KEK)

5.3.6.3. Процес розшифрування (Key Unwrap)

Вхідними даними процесу розшифрування є:

“result” – зашифровані ключові дані,

“dke” – довготривалий ключовий елемент (ДКЕ),

“КЕК” – ключ шифрування ключа (КШК).

Вихідними даними процесу розшифрування є:

“СЕК” – ключові дані (у операції формування “захищені дані” – це ключ шифрування даних КШД).

Процес розшифрування виконується у такі кроки:

а) виконати ініціалізацію алгоритму вхідними даними “КЕК”. Шаблон атрибутів ключа “КЕК” повинен містити як атрибут ДКЕ (SBOX об'єкт);

б) виконати розшифрування “result” на алгоритмі ДСТУ ГОСТ 28147:2009 в режимі гамування із зворотнім зв'язком (GOST28147-CFB), використовуючи ключ “КЕК” та вектор ініціалізації “IV1”:

IV1 = 0x4adda22c79e82105,

TEMP3 = GOST28147-CFB_decrypt(result, IV1, dke, KEK);

г) виконати реверсне перетворення порядку байтів TEMP3, так що перший байт TEMP3 стає останнім байтом, і т.д. Результат перетворення позначимо TEMP2;

д) відокремити складові у TEMP2 (перші 8 байт – це IV, усі інші – це TEMP1):

$$\text{TEMP2} = \text{IV} \parallel \text{TEMP1};$$

е) виконати розшифрування TEMP1 алгоритмом ГОСТ 28147 в режимі гамування із зворотнім зв'язком (GOST28147-CFB), використовуючи ключ “КЕК” та вектор ініціалізації “IV”, отриманий за результатами виконання пункту в):

$$\text{CEKICV} = \text{GOST28147-CFB_decrypt}(\text{TEMP1}, \text{IV}, \text{dke}, \text{KEK});$$

ж) відокремити складові у CEKICV (останні 4 байти – це контрольна сума ICV, усі інші перші – це ключові дані CEK):

$$\text{CEKICV} = \text{CEK} \parallel \text{ICV};$$

з) обчислити контрольну суму (“ICV1”) отриманих ключових даних “CEK” як імітовставку довжини 32 біта (“MAC32”) згідно з розділом 5 ДСТУ ГОСТ 28147:2009.

$$\text{ICV1} = \text{MAC32}(\text{CEK}, \text{dke}, \text{KEK}) [4 \text{ байти}];$$

е) Порівняти отриману за результатами виконання пункту ж) контрольну суму “ICV” з обчисленою “ICV1”. У разі нееквівалентності зазначених контрольних сум, припинити подальше оброблення з результатом “помилка розшифрування ключа”. У разі еквівалентності зазначених контрольних сум, повернути як результат розшифрування алгоритму GOST28147Wrap отримане значення ключового матеріалу “CEK”.

Приклад:

```
CK_SESSION_HANDLE hSession = ...; // handle of session
```

```
CK_BYTE           wrappedKey[1024];
CK_ULONG          ulWrappedKeyLen = sizeof(wrappedKey);
```

```
pMechWrap = (CK_MECHANISM_PTR) calloc(sizeof(CK_MECHANISM), 1);
if (pMechWrap == NULL) printf("Insufficient memory available\n");
```

```
pMechWrap->mechanism      = CKM_GOST28147_KEY_WRAP;
pMechWrap->pParameter      = NULL;
pMechWrap->parameterLen    = 0;
```

```
rv = C_WrapKey(hSession, pMechWrap, hWrappingKey,
```

```

        hWrappedKey, wrappedKey, &ulWrappedKeyLen);
if (rv) printf("\nC_WrapKey failed: %d", rv);

```

```

CK_ATTRIBUTE    TemplateUnWrappedKey[] = {
    {CKA_TOKEN,      &True,      sizeof(CK_BBOOL)},
    {CKA_SENSITIVE,  &False,     sizeof(CK_BBOOL)},
};

```

```

rv = C_UnwrapKey(hSession, pMechWrap, hWrappingKey,
                wrappedKey, ulWrappedKeyLen,
                TemplateUnWrappedKey, NUMITEMS(TemplateUnWrappedKey),
                &hNewKey);
if (rv != CKR_OK) printf("\nC_UnwrapKey failed: %d", rv);
else printf ("\nKey unwrapped. It handle is : %d", hNewKey);

```

5.4. ГОСТ 34.311-95

5.4.1. Визначення

Цей розділ визначає такі механізми:

СКМ_GOST34311 - гешування за ГОСТ 34.311-95.

5.4.2. Опис механізму

Пристрої, що підтримують гешування за ГОСТ 34.311-95, повинні повертати у списку механізмів (функція «C_GetMechanismList()») такі ідентифікатори механізмів:

```

СКМ_GOST34311      = СКМ_VENDOR_DEFINED + 0x00420021;
де
СКМ_VENDOR_DEFINED = 0x80000000.

```

Механізм повинен підтримувати функції гешування як у одно-, так і мульти-частинних операціях (див розділ 4.9).

Інформація про механізм, що повертається структурою СК_МЕCHANISM_INFO, повинна бути такою:

```

ulMinKeySize      = 0;
ulMaxKeySize      = 0;
Flags              = CKF_DIGEST [ | CKF_HW ];

```

Додатковий прапор [CKF_HW] повинен встановлюватися у разі апаратної реалізації.

Параметри алгоритму задаються такою структурою:

```
typedef struct CK_GOST34311_PARAMS {  
    CK_BYTE      sbbox[66];  
    CK_BYTE      iv[32];  
} CK_GOST34311_PARAMS;
```

```
typedef CK_GOST34311_PARAMS* CK_GOST34311_PARAMS_PTR;
```

де поля мають такі значення:

sbbox – DER-кодоване значення ДКЕ як об'єкту SBOX (див. розділ 5.2);
iv – стартовий вектор.

Примітка. Поле sbbox структури CK_GOST34311_PARAMS фактично містить 66 байтів, якщо ДКЕ задається значенням, та менше ніж 66 байтів, якщо ДКЕ задано об'єктним ідентифікатором.

Передбачено такі параметри за замовчуванням:

- ДКЕ №1 Наказу №114;
- нульовий стартовий вектор.

Параметри за замовчуванням застосовуються за відсутності явно вказаних параметрів механізму (NULL).

Довжина вихідних даних механізму – 32 байти.

Приклад:

```
CK_SESSION_HANDLE hSession = ...; // handle of session
```

```
CK_MECHANISM_PTR pMechGost34311;  
CK_GOST34311_PARAMS_PTR pParamGost34311;  
CK_BYTE digestData[32];  
CK_SIZE digestLen = sizeof(digestData);
```

```
CK_BYTE message[] = "Тестове повідомлення";
```

```
static CK_BYTE OID_GOST28147_SBOX_1[14] = {  
    0x06, 0x0C, 0x2A, 0x86, 0x24, 0x02, 0x01, 0x01, 0x01, 0x01,  
    0x01, 0x01, 0x0A, 0x01  
}; //1.2.804.2.1.1.1.1.1.1.10.1
```

```
pParamGost34311 = (CK_GOST34311_PARAMS_PTR)  
    calloc(sizeof(CK_GOST34311_PARAMS), 1);
```

```
memcpy(pParamGost34311->sbbox, OID_GOST28147_SBOX_1,
```

```

        sizeof(OID_GOST28147_SBOX_1));
pParamGost34311->iv = NULL;

pMechGost34311 = (CK_MECHANISM_PTR)
    calloc(sizeof(CK_MECHANISM),1);

pMechGost34311->pParameter = pParamGost34311;
pMechGost34311->parameterLen = sizeof(pParamGost34311);
pMechGost34311->mechanism = CKM_GOST34311;

rv = C_DigestInit(hSession, pMechGost34311);
if (rv != CKR_OK) printf("\nC_DigestInit failed: %08X\n", rv);

rv = C_Digest(hSession, message, sizeof(message)-1,digestData,&digestLen);
if (rv != CKR_OK) printf("\nC_Digest failed: %08X", rv);

```

5.5. ДСТУ 4145-2002

5.5.1. Визначення

Цей розділ визначає ключі типу CKK_DSTU4145, об'єкт EC_PARAMS – параметри еліптичної кривої для ключових, та такі механізми:

формування/ перевіряння ЕЦП без обчислення хеш значення повідомлення (даних);

обчислення/ перевіряння ЕЦП з обчисленням хеш геш значення повідомлення згідно з алгоритмом ГОСТ 34.311-95.

Ключами алгоритму ДСТУ 4145-2002 є ключові пари об'єктів, сесійні чи постійні. Один з них (особистий, або приватний ключ) має клас SKO_PRIVATE_KEY, а інший (відкритий ключ) – клас SKO_PUBLIC_KEY. Атрибут типу SKA_KEY_TYPE для обох ключів повинен мати значення CKK_DSTU4145:

```

CKK_DSTU4145 = CKM_VENDOR_DEFINED + 0x00420131;
де
CKM_VENDOR_DEFINED = 0x80000000.

```

5.5.2. Об'єкт EC_PARAMS

Об'єкт EC_PARAMS зберігає параметри еліптичної кривої для ключових пар, у тому числі ДСТУ 4145-2002 як атрибут SKA_EC_PARAMS або SKA_ECDSA_PARAMS класу даних SKO_DATA.

Об'єкт EC_PARAMS використовується як атрибут ключових об'єктів, у тому числі CKK_DSTU4145, а тому розглядається, як окремий об'єкт даних,

ідентифікатор атрибуту якого визначається відповідно до стандарту PKCS#11 таким чином:

СКА_EC_PARAMS = 0x00000180;

ASN.1 структура EC_PARAMS визначається таким чином:

```
EC_PARAMS ::= CHOICE {  
  ecBinary          ECBinary,          параметри еліптичної кривої  
                                         загального виду  
  namedCurve        OBJECT IDENTIFIER, об'єктний ідентифікатор  
                                         стандартної еліптичної кривої, що  
                                         рекомендована ДСТУ 4145-2002  
}
```

```
ECBinary ::= SEQUENCE {  
  f      BinaryField,          основне поле  
  a      INTEGER (0..1),       коефіцієнт А еліптичної кривої  
  b      OCTET STRING,         коефіцієнт В еліптичної кривої  
  n      INTEGER,              порядок базової точки (додатне  
                               ціле)  
  bp*    OCTET STRING,         базова точка еліптичної кривої  
  cofactor** INTEGER OPTIONAL кофактор  
}
```

```
BinaryField ::= SEQUENCE {  
  m      INTEGER,              ступінь розширення основного  
                               поля  
  CHOICE {  
    Trinomial,                 примітивний тричлен  
    Pentanomial                примітивний п'ятичлен  
  }  
}
```

Trinomial ::= INTEGER

```
Pentanomial ::= SEQUENCE {  
  k INTEGER,  
  j INTEGER,  
  l INTEGER}
```

Рекомендується використовувати namedCurve, а не ecBinary.

Усі параметри еліптичної кривої ДСТУ 4145-2002 представляються у MSB (Big-Endian) форматі.

Примітки:

* має 2 варіанти формату подання:

- у стиснутому форматі (MSB порядок байт), наприклад [x]. При використанні цієї форми подання базової точки, швидкість виконання операції з ДСТУ 4145 сповільнюється на час, необхідний для відновлення до звичайного вигляду;
- у звичайному/ не стиснутому форматі (MSB порядок байт), задаються координати x та y точки, додається перший байт 0x04, як ознака не стиснутого формату, наприклад [0x04, x,y].

** за відсутністю цього параметру, приймається значення за замовчуванням, що дорівнює 2.

Рекомендується використовувати зображення базової точки кривої у звичайному/ не стиснутому форматі. Додатково може використовуватися і стиснутий формат зображення точки.

При створенні об'єкту EC_PARAMS, якщо такий об'єктний ідентифікатор кривої вже існує на токени, повертається помилка CKR_FUNCTION_CANCELED.

При видаленні об'єкту EC_PARAMS з токени, якщо поточний об'єктний ідентифікатор кривої використовується в будь-якому ключовому об'єкті на токени, повертається помилка CKR_FUNCTION_CANCELED.

Для об'єктів EC_PARAMS не передбачено механізму генерації.

Об'єкти EC_PARAMS мають стандартний клас даних CKA_CLASS = SKO_DATA та, крім цього, повинні містити такі обов'язкові атрибути:

Таблиця 5.8. Атрибути, що підтримуються об'єктом EC_PARAMS

Атрибути	Тип даних	Значення
CKA_CLASS ¹	CK_OBJECT_CLASS	Клас об'єкта
CKA_VALUE ^{1,7}	Byte array	Значення параметрів еліптичної кривої, DER-кодоване значення ASN.1 структури параметрів ECBinary
CKA_VALUE_LEN ²	CK_ULONG	Довжина значення атрибута CKA_VALUE в байтах (заповнюється автоматично при створенні об'єкту)
CKA_OBJECT_ID ¹	Byte array	Об'єктний ідентифікатор (namedCurve) параметрів еліптичної кривої, DER-кодоване значення ASN.1 OBJECT IDENTIFIER.

Атрибути	Тип даних	Значення
СКА_LABEL	RFC2279 рядок	Опис об'єкту (за замовчуванням пусто).
СКА_TOKEN	CK_BBOOL	CK_TRUE, якщо об'єкт є об'єктом токена; CK_FALSE якщо об'єкт є сесійним.
СКА_PRIVATE	CK_BBOOL	CK_TRUE, якщо об'єкт є особистим об'єктом; CK_FALSE, якщо об'єкт є відкритим об'єктом.
СКА_SENSITIVE ^{1,11}	CK_BBOOL	CK_TRUE, якщо атрибут об'єкту СКА_VALUE є чутливим (захищений від перегляду)
СКА_EXTRACTABLE ^{1,12}	CK_BBOOL	CK_TRUE, якщо об'єкт є таким, що вилучається, та може бути «обгорнутим»/ зашифрованим
СКА_MODIFIABLE	CK_BBOOL	CK_TRUE, якщо об'єкт може бути змінений

Примітки див. у Таблиці 3.1.

Також:

- Якщо СКА_PRIVATE = CK_FALSE, то об'єкт є видимим для неавторизованого користувача.
- Якщо СКА_SENSITIVE = CK_FALSE, то значення ДКЕ (СКА_VALUE) є видимим (може бути отримано функцією C_GetAttributeValue).
- Якщо СКА_EXTRACTABLE = CK_FALSE, то значення ДКЕ (СКА_VALUE) не може бути згорнуто функцією C_WrapKey.
- Якщо СКА_MODIFIABLE = CK_FALSE, то об'єкт не може бути модифікований (змінений) після його створення.

При використанні цього об'єкту у якості атрибуту інших об'єктів (ключі) чи механізмами, їх відповідний параметр повинен отримати значення об'єктного ідентифікатора (namedsbox) у атрибуті СКА_EC_PARAMS, що співпадає із значенням атрибута СКА_OBJECT_ID об'єкту EC_PARAMS.

Для об'єктів EC_PARAMS не передбачено механізму генерації.

У таблиці 5.9 наведено набір обов'язкових атрибутів шаблону об'єкта EC_PARAMS для створення об'єкта функцією C_CreateObject.

Таблиця 5.9. Обов'язкові атрибути шаблону об'єкта EC_PARAMS (значення наведені у якості зразку і не є обов'язковими)

Атрибути	Значення
СКА_CLASS	СКО_DATA
СКА_VALUE	(DER-кодоване значення ASN.1 структури ECBinary)

Атрибути	Значення
CKA_OBJECT_ID	Об'єктний ідентифікатор параметрів еліптичної кривої
CKA_SENSITIVE	CK_TRUE
CKA_EXTRACTABLE	CK_FALSE

Шаблон може містити додаткові атрибути, які дозволяються для об'єктів класу CKO_DATA.

Приклад створення об'єкту EC_PARAMS:

```
CK_BYTE ec_params[] = {0x30, 0x78, 0x30, 0x07, 0x02, 0x02, 0x01, 0x01,
0x02, 0x01, 0x0C, 0x02, 0x01, 0x00, 0x04, 0x21, 0x01, 0xCE, 0xF4, 0x94, 0x72,
0x01, 0x15, 0x65, 0x7E, 0x18, 0xF9, 0x38, 0xD7, 0xA7, 0x94, 0x23, 0x94,
0xFF, 0x94, 0x25, 0xC1, 0x45, 0x8C, 0x57, 0x86, 0x1F, 0x9E, 0xEA, 0x6A,
0xDB, 0xE3, 0xBE, 0x10, 0x02, 0x21, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x67, 0x59,
0x21, 0x3A, 0xF1, 0x82, 0xE9, 0x87, 0xD3, 0xE1, 0x77, 0x14, 0x90, 0x7D,
0x47, 0x0D, 0x04, 0x21, 0x00, 0x2A, 0x29, 0xEF, 0x20, 0x7D, 0x0E, 0x9B,
0x6C, 0x55, 0xCD, 0x26, 0x0B, 0x30, 0x6C, 0x7E, 0x00, 0x7A, 0xC4, 0x91,
0xCA, 0x1B, 0x10, 0xC6, 0x23, 0x34, 0xA9, 0xE8, 0xDC, 0xD8, 0xD2, 0x0F,
0xB6, 0x02, 0x01, 0x02
};
```

```
CK_BYTE OID[] = {0x06,0x0C,0x2A,0x86,0x24,0x02,0x01,
0x01,0x01,0x01,0x01,0x01,0x02,0x20};
```

```
CK_BYTE          LabelEC_Paparms[]          = "EC_PARAMS";
CK_OBJECT_CLASS  ClassEC_Paparms           = CKO_DATA;
CK_BBOOL         Token                     = True;
CK_ATTRIBUTE     TemplateEC_Paparms[]       = {
    {CKA_LABEL,      LabelEC_Paparms, sizeof(LabelEC_Paparms)-1},
    {CKA_CLASS,      &ClassEC_Paparms, sizeof(ClassEC_Paparms)},
    {CKA_TOKEN,      &Token,          sizeof(CK_BBOOL)},
    {CKA_PRIVATE,    &False,           sizeof(False)},
    {CKA_VALUE,      ec_params,        sizeof(ec_params)},
    {CKA_OBJECT_ID,  OID,               sizeof(OID)},
    {CKA_SENSITIVE,  &False,           sizeof(CK_BBOOL)},
    {CKA_EXTRACTABLE, &False,          sizeof(CK_BBOOL)},
};
```

5.5.3. Ключові об'єкти

Ключові об'єкти – ключова пара ДСТУ4145-2002 є об'єктами класу CKA_CLASS, що повинен мати значення SKO_PUBLIC_KEY для відкритих ключів та значення SKO_PRIVATE_KEY для особистих ключів. Атрибут типу CKA_KEY_TYPE для ключів обох видів повинен мати значення CKK_DSTU4145:

CKK_DSTU4145 = CKK_VENDOR_DEFINED + 0x00420131;

де

CKM_VENDOR_DEFINED = 0x80000000.

А таблиці 5.10 наведено загальний набір атрибутів для особистих та відкритих ключів типу CKK_DSTU4145. Далі, у таблиці 5.11 та 5.12 наведено набори специфічних атрибутів, що повинні підтримуватися відповідно для особистих та відкритих ключів.

Таблиця 5.10. Загальний набір атрибутів ключів ДСТУ 4145-2002

Атрибути	Тип даних	Значення
CKA_CLASS	CK_OBJECT_CLASS	Клас ключа
CKA_KEY_TYPE	CK_KEY_TYPE	Тип ключа
CKA_SBOX ¹	Byte array	DER-кодоване значення ASN.1 структури об'єкта SBOX (див. розділ 5.2)
CKA_EC_PARAMS	Byte array	DER-кодоване значення параметрів – ASN.1 структури EC_PARAMS
CKA_KEY_SIZE	CK_ULONG	Довжина ключа у бітах. (заповнюється автоматично при генерації об'єкта)
CKA_LABEL	RFC2279 рядок	Опис об'єкту (за замовчуванням пусто).
CKA_ID ⁹	Byte array	Унікальний ідентифікатор ключової пари (схема виробництва визначається виробником)
CKA_DERIVE	CK_BBOOL	CK_TRUE, якщо ключ підтримує виробництво ключа – derivation, тобто, якщо інші ключі можуть бути отримані від цього (за замовченням CK_FALSE)
CKA_TOKEN	CK_BBOOL	CK_TRUE, якщо об'єкт є об'єктом токена;

Атрибути	Тип даних	Значення
		CK_FALSE якщо об'єкт є сесійним.
CKA_PRIVATE	CK_BBOOL	CK_TRUE, якщо об'єкт є особистим об'єктом; CK_FALSE, якщо об'єкт є відкритим об'єктом.
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE, якщо об'єкт може бути змінений. Значення задається при створенні об'єкту та не може бути змінено після створення
CKA_LOCAL ^{2,4,6}	CK_BBOOL	CK_TRUE, тільки якщо ключ був або згенеровано локально (тобто, у токени) шляхом виклику C_GenerateKey або C_GenerateKeyPair або створено шляхом виклику C_CopyObject, як копія ключа, який мав атрибут CKA_LOCAL, що встановлено у CK_TRUE

Примітки див. у Таблиці 3.1.

Також:

- Значення атрибуту CKA_ID генерується автоматично для особистого ключа та копіюється до відповідного атрибуту відкритого ключа, алгоритм формування визначається виробником токenu.
- Відповідне значення атрибутів CKA_KEY_SIZE заповнюється або автоматично при генерації об'єкту, або задається через шаблони особистого та відкритого ключа.
- Відповідні значення атрибутів CKA_EC_PARAMS та CKA_SBOX копіюються із шаблону відкритого ключа до шаблону особистого ключа, якщо.вино не задані у шаблоні особистого ключа.

Відкриті ключі, окрім атрибутів класу CKA_CLASS та типу CKA_KEY_TYPE, повинні мати такі атрибути:

Таблиця 5.11. Специфічні атрибути, що підтримуються особистими ключами ДСТУ 4145-2002

Атрибут	Тип даних	Значення
CKA_VALUE	Big integer	Значення особистого ключа (довге ціле відповідно до ДСТУ 4145-2002)

Атрибут	Тип даних	Значення
СКА_VALUE_LEN ²	CK_ULONG	Довжина значення атрибута СКА_VALUE в байтах (заповнюється авто-матично при створенні об'єкту)
СКА_SENSITIVE ^{1,11}	CK_BBOOL	CK_TRUE, якщо атрибут об'єкту СКА_VALUE є чутливий (захищений від перегляду)
СКА_EXTRACTABLE ^{1,12}	CK_BBOOL	CK_TRUE, якщо ключ є таким, що вилучається, та може бути «обгорнутим»/ зашифрованим
СКА_SIGN	CK_BBOOL	CK_TRUE, якщо ключ підтримує підписування, де підпис є додатком до даних

Примітки див. у Таблиці 3.1.

Таблиця 5.12. Специфічні атрибути, що підтримуються відкритими ключами ДСТУ 4145-2002

Атрибут	Тип даних	Значення
СКА_EC_POINT	Byte array	DER-кодоване представлення відкритого ключа, як ASN.1 OCTET STRING, яке містить зображення точки еліптичної кривої
СКА_VERIFY	CK_BBOOL	CK_TRUE, якщо ключ підтримує верифікацію (перевіряння) підпису, де підпис додається до даних

Примітки див. у Таблиці 3.1.

Зображення точки відкритого ключа може бути у таких форматах:

- у стиснутому форматі (MSB порядок байт), наприклад [x]. При використанні цієї форми подання базової точки, швидкість виконання операції з ДСТУ 4145 сповільнюється на час, необхідний для відновлення до звичайного вигляду;
- у звичайному/ не стиснутому форматі (MSB порядок байт), задаються координати x та y точки, додається перший байт 0x04, як ознака не стиснутого формату, наприклад [0x04, x,y].

Рекомендується використовувати зображення точки відкритого ключа у звичайному/ не стиснутому форматі. Додатково може використовуватися і стистутий формат зображення точки.

Для створення ключової пари ДСТУ 4145-2002 передбачено механізм генерації СКМ_DSTU4145_KEY_PAIR_GEN. Також створення ключів ДСТУ 4145-2002 можливе за допомогою функції C_CreateObject. У таблицях 5.13 та 5.14

наведено набори обов'язкових атрибутів шаблонів для створення відповідно особистого та відкритого ключа.

Таблиця 5.13. Обов'язкові атрибути шаблону особистого ключа ДСТУ 4145-2002 (значення наведені у якості зразку і не є обов'язковими)

Атрибути	Значення
СКА_CLASS	CKO_PRIVATE_KEY
СКА_KEY_TYPE	CKK_DSTU4145
СКА_SBOX	OID_GOST28147_SBOX_1 (ДКЕ №1 Наказу №114)
СКА_EC_PARAMS	OID_DSTU4145_POLY_CURVE_191
СКА_VALUE	Значення особистого ключа
СКА_ID	Унікальний ідентифікатор ключової пари
СКА_KEY_SIZE	191

Таблиця 5.14. Обов'язкові атрибути шаблону відкритого ключа ДСТУ 4145-2002 (значення наведені у якості зразку і не є обов'язковими)

Атрибути	Значення
СКА_CLASS	CKO_PRIVATE_KEY
СКА_KEY_TYPE	CKK_DSTU4145
СКА_SBOX	OID_GOST28147_SBOX_1 (ДКЕ №1 Наказу №114)
СКА_EC_PARAMS	OID_DSTU4145_POLY_CURVE_191
СКА_EC_POINT	Значення відкритого ключа
СКА_ID	Унікальний ідентифікатор ключової пари
СКА_KEY_SIZE	191

Для підписування/перевіряння підпису слід встановлювати флаги СКА_SIGN = TRUE для особистого ключа та СКА_VERIFY = TRUE для відкритого ключа; для ключової пари шифрування (Діффі-Геллмана) слід задавати такі параметри: СКА_DERIVE = CK_TRUE.

Атрибути СКА_SIGN_RECOVER, СКА_VERIFY_RECOVER, СКА_WRAP, СКА_UNWRAP, СКА_ENCRYPT та СКА_DECRYPT ігноруються, оскільки ці операції не підтримуються базовим алгоритмом.

До ключових об'єктів CKK_DSTU4145 повинні застосовуватися, щонайменше, такі функції: C_CreateObject, C_DestroyObject, C_GetAttributeValue, C_SetAttributeValue, C_FindObjects, C_FindObjectsFinal.

Приклад (створення постійного об'єкту - відкритий ключ на токени):

```

CK_SESSION_HANDLE hSession = ...;

CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_DSTU4145;
CK_UTF8CHAR label[] = "Dstu 4145 public key object";
CK_BYTE ecParams[] = {...};
CK_BYTE ecPoint[] = {...};
CK_BYTE sbox[] = {...};
CK_BYTE id[] = {...};
CK_ULONG keySize = 191;
CK_BBOOL true = CK_TRUE;

CK_OBJECT_HANDLE hKey = NULL;

CK_ATTRIBUTE template[] = {
    {CKA_CLASS, &class, sizeof(class)},
    {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
    {CKA_SBOX, sbox, sizeof(sbox)},
    {CKA_EC_PARAMS, ecParams, sizeof(ecParams)},
    {CKA_EC_POINT, ecPoint, sizeof(ecPoint)},
    {CKA_ID, id, sizeof(id)},
    {CKA_KEY_SIZE, &keySize, sizeof(keySize)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
};

/* Create DSTU4145 public key */
rv = C_CreateObject(hSession, template, NUMITEMS(template), &hKey);
if (rv == CKR_OK) {
    .
    .
}

```

5.5.4. Прапори інформації про механізми з еліптичними кривими

Інформація про механізми («CK_MECHANISM_INFO») з еліптичними кривими додатково до прапорів, визначених у розділі 2.7.5, може містити такі прапори:

Таблиця 5.15. Прапори інформації про механізми

Прапор	Значення	Опис
CKF_EC_F_2M	0x00200000	True, якщо механізм може використовуватися з параметрами еліптичних кривих (EC) у полях F_{2^m}

Прапор	Значення	Опис
CKF_EC_ECPARAMETERS	0x00400000	True, якщо механізм може використовуватися з параметрами еліптичних кривих (EC), які задаються явно як ecParameters (ECBinary, див. об'єкт EC_PARAMS)
CKF_EC_NAMEDCURVE	0x00800000	True, якщо механізм може використовуватися з параметрами еліптичних кривих (EC), які задаються як об'єктний ідентифікатор кривої (namedCurve, див. об'єкт EC_PARAMS)
CKF_EC_UNCOMPRESS	0x01000000	True, якщо механізм може використовуватися з точками кривої, поданими у звичайному/ не стиснутому форматі.
CKF_EC_COMPRESS	0x02000000	True, якщо механізм може використовуватися з точками кривої, поданими у стиснутому форматі.

5.5.5. Механізми генерації ключів

Механізм генерації CKM_DSTU4145_KEY_PAIR_GEN описує генерацію ключової пари відповідно до Розділу 9 ДСТУ 4145-2002. Механізм визначається таким ідентифікатором:

CKM_DSTU4145_KEY_PAIR_GEN = CKM_VENDOR_DEFINED + 0x00420042;

де

CKM_VENDOR_DEFINED = 0x80000000.

Інформація про механізм («CK_MECHANISM_INFO») повинна бути такою:

ulMinKeySize = 163;

ulMaxKeySize = 509;

Flags = CKF_GENERATE_KEY_PAIR | CKF_EC_F_2M
| CKF_EC_ECPARAMETERS | CKF_EC_NAMEDCURVE
| CKF_EC_UNCOMPRESS
[CKF_EC_COMPRESS] [CKF_HW];

Додатковий прапор [CKF_HW] повинен встановлюватися у разі апаратної реалізації.

Необов'язкові параметри механізму задаються такою структурою (аналогічно ключам CKK_GOST28147, розділ 5.3.3):

```
typedef struct CK_SEED_PARAMS {
    CK_BYTE    seed[64];
} CK_SEED_PARAMS;
typedef CK_SEED_PARAMS * CK_SEED_PARAMS_PTR;
```

де поля мають такі значення:

seed – початковий матеріал (зерно) для генератора випадкових чисел.

Якщо застосовується механізм без параметрів (NULL), то початкове значення генератора випадкових чисел може бути встановлено окремим викликом функції C_SeedRandom, що передує механізму генерації.

Ключові пари створюються функцією C_GenerateKeyPair за допомогою цього механізму. Шаблони атрибутів ключової пари за змовченням визначено у Таблиці 5.16 та у Таблиці 5.17.

Таблиця 5.16. Атрибути особистих ключів за замовчуванням механізму CKM_DSTU4145_GENERATE_KEY_PAIR.

Атрибут	Значення
CKA_LABEL	“Dstu 4145 Private Key”
CKA_KEY_TYPE ¹	CKK_DSTU4145
CKA_CLASS ¹	CKO_PRIVATE_KEY
CKA_EC_PARAMS ¹	OID_DSTU4145_POLY_CURVE_191
CKA_SBOX ¹	OID_GOST28147_SBOX_1 (ДКЕ №1 Наказу №114)
CKA_VALUE ¹	Випадкова послідовність
CKA_ID ⁹	*залежить від CKA_VALUE
CKA_KEY_SIZE ⁹	191
CKA_TOKEN ¹	CK_FALSE
CKA_SIGN ¹	CK_TRUE
CKA_PRIVATE ¹	CK_TRUE
CKA_SENSITIVE ^{1,11}	CK_TRUE
CKA_EXTRACTABLE ^{1,12}	CK_FALSE
CKA_DERIVE	CK_FALSE
CKA_LOCAL	CK_TRUE
CKA_MODIFIABLE	CK_TRUE

Примітки див. у Таблиці 3.1.

Таблиця 5.17 Атрибути відкритих ключів за замовчуванням механізму CKM_DSTU4145_GENERATE_KEY_PAIR.

Атрибут	Значення
---------	----------

Атрибут	Значення
СКА_LABEL	“Dstu 4145 Public Key”
СКА_KEY_TYPE ¹	CKK_DSTU4145
СКА_CLASS ¹	CKO_PUBLIC_KEY
СКА_EC_PARAMS ¹	копіюється із особистого ключа
СКА_SBOX ¹	копіюється із особистого ключа
СКА_EC_POINT ¹	обчислене значення відкритого ключа
СКА_ID ⁹	копіюється із особистого ключа
СКА_KEY_SIZE ⁹	191
СКА_TOKEN ¹	CK_FALSE
СКА_VERIFY ¹	CK_TRUE
СКА_PRIVATE	CK_FALSE
СКА_DERIVE	CK_FALSE
СКА_LOCAL	CK_TRUE
СКА_MODIFIABLE	CK_TRUE

Примітки див. у Таблиці 3.1.

Слід зазначити, що атрибути СКА_EC_PARAMS та СКА_SBOX достатньо задавати для шаблону відкритого ключа, тоді параметри для особистого ключа повинні братися з шаблону відкритого ключа. Атрибут СКА_ID та СКА_KEY_SIZE генерується виробником та є ідентичним для особистого і відкритого ключів.

Оскільки типи ключів, що генеруються, неявно визначаються у механізмі генерації ключової пари, то у шаблоні не обов’язково застосовувати тип ключа. Якщо один із шаблонів застосовує тип ключа, який є несумісним з механізмом генерації, то C_GenerateKeyPair не виконується і повертається код помилки CKR_TEMPLATE_INCONSISTENT. Атрибут СКА_CLASS оброблюється подібним чином.

Приклад:

```
CK_SESSION_HANDLE hSession = ...;
```

```
CK_OBJECT_HANDLE hPublicKey, hPrivateKey;
```

```
CK_MECHANISM mechanism = {
    CKM_DSTU4145_KEY_PAIR_GEN, NULL_PTR, 0
};
```

```
CK_OBJECT_CLASS classPubKey = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_DSTU4145;
CK_UTF8CHAR labelPubKey [] = “Dstu 4145 public key object”;
CK_UTF8CHAR labelPrivKey [] = “Dstu 4145 private key object”;
CK_BYTE ecParams[] = {...};
```

```

CK_BYTE ecPoint[] = {...};
CK_BYTE sbox[] = {...};
CK_BBOOL true = CK_TRUE;

CK_ATTRIBUTE publicKeyTemplate [] = {
    {CKA_CLASS, & classPubKey, sizeof(classPubKey)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, labelPrivKey, sizeof(labelPrivKey)-1},
    {CKA_EC_PARAMS, ecParams, sizeof(ecParams)},
    {CKA_SBOX, sbox, sizeof(sbox)}
};

CK_ATTRIBUTE privateKeyTemplate[] = {
    {CKA_CLASS, & classPubKey, sizeof(classPubKey)},
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, labelPrivKey, sizeof(labelPrivKey)-1}
};

rv = C_GenerateKeyPair(
    hSession, &mechanism,
    publicKeyTemplate, 5,
    privateKeyTemplate, 3,
    &hPublicKey, &hPrivateKey);
if (rv == CKR_OK) {
    .
    .
}

```

5.5.6. Механізми ДСТУ 4145-2002 без хешування

Механізм ДСТУ 4145-2002 без хешування (без обчислення хеш значення) задається таким ідентифікатором:

CKM_DSTU4145 = CKM_VENDOR_DEFINED + 0x00420031;

де

CKM_VENDOR_DEFINED = 0x80000000.

Механізм CKM_DSTU4145 є механізмом для одно-частинного (Single-part) підписування (див. функцію C_Sign) та перевіряння підпису (див. функцію C_Verify) ДСТУ 4145-2002; механізм лише обробляє хеш значення, але не обчислює його.

Обмеження на розмір вхідних буферів/ даних залежить від реалізації криптографічного пристрою. У разі нехватки пам'яті на пристрої для приймання вхідних даних буде повернено помилки CKR_DATA_LEN_RANGE.

Обмеження на типи ключів та довжину вхідних/вихідних даних наведено у таблиці 5.18:

Таблиця 5.18. Тип ключа та довжини даних

Функція	Тип ключа	Довжина вхідних даних	Довжина вихідних даних
C_Sign	CKK_DSTU4145 private key	any ²	nLen
C_Verify	CKK_DSTU4145 public key	any ² , ≤nLen ¹	N/A

¹ Довжина підпису.

² Довжина хеш значення.

Пристрої, що підтримують формування та перевіряння підпису за ДСТУ 4145-2002, повинні повертати у списку механізмів (функція «C_GetMechanismList()») ідентифікатор цього алгоритму. Інформація про механізм CK_MECHANISM_INFO повинна бути такою:

```
ulMinKeySize    = 163;
ulMaxKeySize    = 509;
Flags           = CKF_SIGN | CKF_VERIFY | CKF_EC_F_2M
                 | CKF_EC_ECPARAMETERS | CKF_EC_NAMEDCURVE
                 | CKF_EC_COMPRESS
                 [| CKF_EC_UNCOMPRESS] [| CKF_HW];
```

Додатковий прапор [CKF_HW] повинен встановлюватися у разі апаратної реалізації.

Цей механізм може мати параметри CK_SEED_PARAMS для встановлення початкового значення генератора випадкових чисел при обчисленні перед-підпису, згідно з ДСТУ 4145-2002, функції підписування (C_Sign).

Якщо застосовується механізм без параметрів (NULL), то початкове значення генератора випадкових чисел може бути встановлено окремим викликом функції C_SeedRandom, що передуює механізму підписування.

Приклад використання механізму CKM_DSTU4145:

```
#define GOST34311_DIGEST_LENGTH 32
```

```
CK_OBJECT_HANDLE hPublicKey, hPrivateKey;
CK_MECHANISM mech = {CKM_DSTU4145_KEY_PAIR_GEN, NULL, 0};
```

```
rv = C_GenerateKeyPair (hSession, &mech,
NULL, 0, NULL, 0, &hPublicKey, &hPrivateKey);
if (rv != CKR_OK) {
    printf("\nC_GenerateKeyPair failed: %d", rv);
    return;
```

```

    }

    CK_BYTE    TextData[] = "This sample will be hashed and signed";
    CK_BYTE    DigestData[GOST34311_DIGEST_LENGTH];
    CK_SIZE    digestLen = GOST34311_DIGEST_LENGTH;

    mech.mechanism = CKM_GOST34311;

    rv = C_DigestInit(hSession, &mech);
    if (rv != CKR_OK) {
        printf("\nC_DigestInit failed: %08X\n", rv);
        return;
    }
    rv = C_Digest(hSession, TextData, sizeof(TextData), DigestData, &digestLen);
    if (rv != CKR_OK) {
        printf("\nC_Digest failed: %08X\n", rv);
        return;
    }

    CK_BYTE_PTR pSignData;
    CK_SIZE     SignLen;

    mech.mechanism = CKM_DSTU4145;
    rv = C_SignInit(hSession, &mech, hPrivateKey);
    if (rv != CKR_OK) {
        printf("\nC_SignInit failed: %d", rv);
        return;
    }

    rv = C_Sign(hSession, DigestData, sizeof(DigestData), NULL, &SignLen);
    if (rv != CKR_OK) {
        printf("\nC_Sign failed: %d", rv);
        return;
    }
    pSignData = (CK_BYTE_PTR)calloc(SignLen,1);
    if (pSignData == NULL) {
        printf("\nNot enough memory");
        return;
    }
    rv = C_Sign(hSession, DigestData, sizeof(DigestData), pSignData, &SignLen);
    if (rv != CKR_OK) {
        printf("\nC_Sign failed: %d", rv);
        return;
    }

```

5.5.7. Механізм ДСТУ 4145-2002 з хешуванням

Механізм ДСТУ 4145-2002 з хешуванням за ГОСТ 34.311-95 задається таким ідентифікатором:

CKM_DSTU4145_WITH_GOST34311 = CKM_VENDOR_DEFINED + 0x00420032;

де

CKM_VENDOR_DEFINED = 0x80000000.

Механізм CKM_DSTU4145_WITH_GOST34311 є механізмом як одно- (Single-part), так і багато частинного (Multy-part) підписування (див. функцію C_Sign та C_SignUpdate) та перевіряння підпису (див. функцію C_Verify та C_VerifyUpdate) ДСТУ 4145-2002; механізм обробляє дані, обчислюючи хеш значення, а потім підписування/перевіряння підпису.

Пристрої, що підтримують формування та перевіряння підпису за ДСТУ 4145-2002, повинні повертати у списку механізмів (функція «C_GetMechanismList()») ідентифікатор цього алгоритму. Інформація про механізм («CK_MECHANISM_INFO») повинна бути такою:

ulMinKeySize = 163;
ulMaxKeySize = 509;
Flags = CKF_SIGN | CKF_VERIFY | CKF_EC_F_2M |
CKF_EC_ECPARAMETERS |
CKF_EC_NAMEDCURVE | CKF_EC_COMPRESS;

Обмеження на розмір вхідних буферів/ даних залежить від реалізації криптографічного пристрою. У разі нехватки пам'яті на пристрої для приймання вхідних даних буде повернено помилки CKR_DATA_LEN_RANGE.

Обмеження на типи ключів та довжину вхідних/вихідних даних наведено у таблиці 5.19:

Таблиця 5.19. Тип ключа та довжини даних

Функція	Тип ключа	Довжина вхідних даних	Довжина вихідних даних
C_Sign, C_SignUpdate	CKK_DSTU4145 private key	any ²	nLen
C_Verify, C_VerifyUpdate	CKK_DSTU4145 public key	any ² , ≤nLen ¹	N/A

¹ Довжина підпису.

² Довжина даних, не більше 32 Кбайт.

Цей механізм може мати параметри CK_SEED_PARAMS для встановлення початкового значення генератора випадкових чисел при обчисленні передпідпису, згідно з ДСТУ 4145-2002, функції підписування (C_Sign).

Якщо застосовується механізм без параметрів (NULL), то початкове значення генератора випадкових чисел може бути встановлено окремим викликом функції C_SeedRandom, що передуює механізму підписування.

Подальше формування підпису здійснюється штатним чином з використанням функцій «C_Sign()» та «C_SignUpdate()» / «C_SignFinal()», а перевіряння підпису – з використанням функцій «C_Verify()» та «C_VerifyUpdate()» / «C_VerifyFinal()».

Приклад використання механізму CKM_DSTU4145_WITH_GOST34311:

```
CK_OBJECT_HANDLE hPublicKey, hPrivateKey;  
CK_MECHANISM mech = {CKM_DSTU4145_KEY_PAIR_GEN, NULL, 0};
```

```
rv = C_GenerateKeyPair (hSession, &mech,  
NULL, 0, NULL, 0, &hPublicKey, &hPrivateKey);  
if (rv != CKR_OK) {  
    printf("\nC_GenerateKeyPair failed: %d", rv);  
    return;  
}
```

```
CK_BYTE          TextData[] = "This sample will be signed";
```

```
CK_BYTE_PTR      pSignData;  
CK_SIZE          SignLen;
```

```
mech.mechanism = CKM_DSTU4145_WITH_GOST34311;  
rv = C_SignInit (hSession, &mech, hPrivateKey);  
if (rv != CKR_OK) {  
    printf("\nC_SignInit failed: %d", rv);  
    return;  
}  
rv = C_Sign (hSession, TextData, sizeof(TextData), NULL, &SignLen);  
if (rv != CKR_OK) {  
    printf("\nC_Sign failed: %d", rv);  
    return;  
}  
pSignData = (CK_BYTE_PTR)calloc(SignLen,1);  
if (pSignData == NULL) {  
    printf("\nNot enough memory");  
    return;  
}
```

```

rv = C_Sign (hSession, TextData, sizeof(TextData), pSignData, &SignLen);
if (rv != CKR_OK) {
    printf("\nC_Sign failed: %d", rv);
    return;
}

```

5.6. ECDH Діффі-Хеллмана механізм

5.6.1. Визначення

У цьому розділі визначаються Діффі-Хеллмана (Diffie-Hellman) механізм, який є механізмом виробництва ключа (Key Derivation Mechanism, KDF) шифрування ключів (КШК), заснований на Діффі-Хеллмана схемі узгодження ключів (Key Agreement Scheme) в групі точок еліптичної кривої, як це визначено в міжнародних рекомендаціях RFC 3278, RFC 5008 та національному стандарті ДСТУ ISO/IEC 15946-3:2002 (Додаток А.3. Функція формування ключа ANSI X9.63), де кожна з двох сторін використовує одну пару ключів, що мають однаковий доменні ЕС параметри.

Визначаються такі механізми ECDH (Elliptic Curve Diffie-Hellman):

CKM_DSTU4145_ECDH_COFACTOR_DERIVE;

CKM_DSTU4145_ECDH_DERIVE;

ЕС параметри механізмів:

CK_DSTU4145_ECDH_DERIVE_PARAMS,

та KDF-функції виробництва ключа:

CKD_GOST34311_KDF

Механізм CKM_DSTU4145_ECDH_DERIVE є опціональним.

Механізми використовуються функцією C_DeriveKey.

5.6.2. Параметри механізмів ECDH

ЕС параметри механізмів ECDH позначаються таким чином:

CK_DSTU4145_ECDH_DERIVE_PARAMS

Параметри CK_DSTU4145_ECDH_DERIVE_PARAMS задаються такою структурою:

```

typedef struct CK_DSTU4145_ECDH_DERIVE_PARAMS {
    CK_EC_KDF_TYPE    kdf;
    CK_BYTE            pSharedData[64];
    CK_ULONG           ulSharedDataLen;
}

```

```

        CK_BYTE                pPublicData[131];
    } CK_DSTU4145_ECDH_DERIVE_PARAMS;

```

```

typedef CK_DSTU4145_ECDH_DERIVE_PARAMS*
CK_DSTU4145_ECDH_DERIVE_PARAMS_PTR;

```

де поля мають такі значення:

kdf – визначає тип функції виробництва ключів. Поле може приймати наступні значення:

```
CKD_GOST34311_KDF;
```

pSharedData – байт-масив одноразових (випадкових) даних, які використовуються при генерації КШК (User Keying Material);

ulSharedDataLen – довжина в байтах одноразових (випадкових) даних. Не допускається відсутність таких даних;

pPublicData – DER-кодоване значення відкритого ключа ДСТУ 4145-2002 у форматі EC_POINT (див. розділ 5.5.3), що використовується для формування ключа КШК.

З функцією CKD_GOST34311_KDF:

pSharedData повинна поставляти деякі одноразові дані, що розподілені (відомі) обом сторонам. В іншому випадку, pSharedData повинно повернути код помилки CKR_ARGUMENTS_BAD.

CK_DSTU4145_ECDH_DERIVE_PARAMS_PTR є покажчик на CK_DSTU4145_ECDH_DERIVE_PARAMS.

5.6.3. Тип KDF-функцій

Типи KDF-функцій визначаються як:

```
CK_EC_KDF_TYPE, CK_EC_KDF_TYPE_PTR,
```

де

CK_EC_KDF_TYPE – тип KDF, що використовується для визначення KDF-функції, яка застосовується для отримання КШК із спільного секрету, що виробляється схемою узгодження ключів.

Тип KDF визначається таким чином:

```
typedef CK_ULONG CK_EC_KDF_TYPE;
```

Ідентифікатор KDF-функції:

```
CKD_GOST34311_KDF = CKD_VENDOR_DEFINED + 0x00420211,
```

де

CKD_VENDOR_DEFINED = 0x80000000

5.6.4. Алгоритм KDF

KDF-функція призначається для формування ключового матеріалу, а саме КШК, із обчисленого спільного секрету ZZ, та виконується відповідно до алгоритму:

$KM = H (ZZ \parallel counter \parallel SharedInfo)$

де

KM – ключовий матеріал (рядок байтів), довжина якого залежить від алгоритму H;

H – геш-функція, визначається ідентифікатором механізму ECDH; для типу CKD_GOST34311_KDF використовується ГОСТ 34.311-95 алгоритм;

ZZ – спільний секрет, відповідно до Діффі-Хеллмана схеми узгодження ключів;

counter – це 32-х бітне число, яке представлено як бітовий вектор (чотири байти), записаного у зворотному порядку. Початковим значенням “counter” є 1 (одиниця) для будь-якого ZZ, а його бітовий вектор є “00 00 00 01” (hex); значення counter збільшується на одиницю (incremented) з кожним циклом виконання функції формування ключового матеріалу KM під час генерації ключа КШК;

SharedInfo – DER-кодована ASN.1 структура;

|| – означає операцію конкатенації.

SharedInfo має таку ASN.1 структуру:

```
SharedInfo ::= SEQUENCE {  
    keyInfo AlgorithmIdentifier,  
    entityUInfo [0] EXPLICIT OCTET STRING OPTIONAL,  
    suppPubInfo [2] EXPLICIT OCTET STRING  
}
```

Поля структури “SharedInfo”:

“algorithm” – є ідентифікатор алгоритму ключа шифрування ключа KeyWrapAlgorithm), на якому повинен бути зашифрований ключ шифрування повідомлення (даних). Параметри алгоритму повинні бути NULL (ASN.1 NULL);

“entityUInfo” – це випадковий рядок, який генерує відправник. У повідомленні CMS це значення розміщується в полі “ukm” (“UserKeyingMaterial”) (закодоване як OCTET STRING) структури “KeyAgreeRecipientInfo”. Довжина “entityUInfo” повинна бути 512 біт (64 байти);

“suppPubInfo” - це довжина сформованого ключа КШК в бітах, представлена як бітовий вектор (чотири байти) 32-бітного числа, записаного у

зворотному порядку. Наприклад, довжина ключа в 192 біти повинна бути представлена як бітовий вектор “00 00 00 C0” (hex).

Так як довжина ключового матеріалу КМ залежить від алгоритму Н (тобто ГОСТ 34.311, і становить 256 біт), та може бути не рівною довжині ключа КШК (як рядка байтів), то для отримання КШК із ключового матеріалу КМ використовується такий алгоритм:

А) Якщо довжина КМ більше, ніж довжина КШК, то за КШК приймають перші N байтів КМ, де N – довжина КШК. Випадок має місце, коли у якості WRAP механізму використовується механізм, наприклад, AES128 з довжиною ключа 128 біт (RFC 3394).

Б) Якщо довжина КМ дорівнює довжині КШК, то за КШК приймають КМ. Випадок має місце, коли у якості WRAP механізму використовується механізм, наприклад, ДСТУ ГОСТ 28147:2009 або AES256 з довжиною ключа 256 біт (RFC 3394).

В) Якщо довжина КМ менше, ніж довжина КШК, то:

Встановлюється значення counter = 1, та формується КМ1.

Встановлюється значення counter = 2, та виконується КМ2.

.....

(кількість кроків формування КМі визначається необхідною довжиною ключа КШК).

Отримані блоки ключового матеріалу об'єднуються (concatenation) для отримання необхідної довжини (останні байти останнього блоку КМі відкидаються):

$КМ = КМ1 \parallel КМ2 \parallel \dots$

Таким чином, що довжина КМ повинна бути рівною довжині ключа КШК.

Цей випадок має місце, коли у якості WRAP механізму використовується механізм, наприклад, AES512 з довжиною ключа 512 біт (RFC 3394).

Примітки.

1. Щодо опціональності параметру entityUInfo: Якщо необов'язковий параметр entityUInfo не буде використовуватися, то у випадках А та Б для різних повідомлень буде формуватися один і той же ключ шифрування КШК. Для того щоб уникнути цього, у разі статичного механізму вимагається, а у разі динамічного механізму рекомендується, генерувати випадкове значення entityUInfo для кожного повідомлення, та використовувати його під час формування КШК.

2. Для обчислення хеш-значення за ГОСТ 34.311-95, що використовується під час вироблення КШК, ДКЕ необхідно брати з атрибуту СКА_SBOX об'єкту відкритого ключа одержувача (на стороні відправника) чи об'єкту особистого ключа одержувача hBaseKey (на стороні одержувача). У разі відсутності ДКЕ, використовується ДКЕ №1 Наказу №114, як ДКЕ за замовчуванням.

5.6.5. Обчисленого спільного секрету ZZ

Обчислення спільного секрету не залежить від механізму узгодження ключів (статичний чи динамічний).

Обчислення спільного секрету відправником виконується на основі:

- особистого ключа відправника ($hBaseKey$), та
- відкритого ключа одержувача ($pPublicData$ в параметрах механізмів ECDH).

Обчислення спільного секрету одержувачем виконується на основі:

- особистого ключа одержувача ($hBaseKey$), та
- відкритого ключа відправника ($pPublicData$ в параметрах механізмів ECDH).

Процедура обчислення спільного секрету у цих специфікаціях називається ZZ-функція. Результатом обчислення спільного секрету є ZZ спільний секрет, як елемент поля, в якому виконується обчислення.

Обчислення спільного секрету в групі точок еліптичної кривої (ECDH) з кофакторним множенням в групі ґрунтується на національному стандарті ДСТУ ISO/IEC 15946-3:2006 та виконується наступним чином:

Відправником обчислюється значення точки K еліптичної кривої з координатами (x_k, y_k) як:

$$K = da * h * Pb,$$

а отримувачем –

$$K = db * h * Pa,$$

де

da – особистий ключ відправника,

Pa – відкритий ключ відправника,

db – особистий ключ одержувача,

Pb – відкритий ключ одержувача,

h – кофактор.

Спільним секретом є координата x_k точки K :

$$Z = x_k,$$

де Z – спільний секрет, як елемент поля, в якому виконується обчислення.

У разі обчислення спільного секрету без кофакторного множення значення кофактору приймають рівним одиниці ($h=1$).

Виконання операцій над точками еліптичної кривої, зображення даних, перевірка правильності загальних параметрів алгоритму та правильності ключів здійснюється згідно з національним стандартом України ДСТУ 4145-2002.

Перетворення елемента поля Z на рядок байтів ZZ (Field-Element-to-Octet-String Conversion), що використовується у функціях формування ключа (KDF-функціях), виконується таким чином:

1. Виконати перетворення елемента поля Z на додатне ціле число, як визначено у пункті 5.8 ДСТУ 4145-2002. Позначимо ціле число від Z як ZI .
2. Виконати перетворення цілого ZI на рядок байтів ZZ у форматі Big-Endian.

При зворотному розміщенні байт (Big-Endian) старший байт (big-end) повинен зберігатися за найменшою адресою (як байт з найменшим індексом байт-масиву).

5.6.6. Механізми ECDH

Ці механізми виробляють значення ключа шифрування ключів КШК, відповідно до `СКА_KEY_TYPE` шаблону атрибутів КШК.

Механізм використовується функцією `C_DeriveKey`, що повертає як результат показник на новий ключ КШК, який має атрибути, відповідно до заданого шаблону атрибутів (див. розділ 4.12.5).

Ці механізми мають такі правила щодо атрибутів ключа:

- `СКА_SENSITIVE` і `СКА_EXTRACTABLE` атрибути в шаблоні створеного нового ключа можуть бути зазначені як `CK_TRUE` або `CK_FALSE`. Якщо не вказано у шаблоні, то ці атрибути приймають значення шаблону за замовчуванням.

- Якщо базовий ключ (особистий ключ відправника/ одержувача) має свій `СКА_ALWAYS_SENSITIVE` атрибут, встановлений в `CK_FALSE`, то створений новий ключ буде мати такий же. Якщо базовий ключ має свій `СКА_ALWAYS_SENSITIVE` атрибут, встановлений в `CK_TRUE`, то створений новий ключ атрибут `СКА_ALWAYS_SENSITIVE` того ж значення, як його `СКА_SENSITIVE` атрибут.

Механізм `CKM_DSTU4145_ECDH_DERIVE` є алгоритмом, що містить обчислення спільного секрету ZZ без кофакторного множення, об'єктний ідентифікатор якого такий:

```
id-dhSinglePass-stdDH-gost34311kdf-scheme OBJECT IDENTIFIER ::= { iso(1) member-body(2) Ukraine(804) root(2) security(1) cryptography(1) pki(1) pki-alg(1) pki-alg-asym(3) dhSinglePassstdDH-gost34311kdf(5) }
```

Механізм CKM_DSTU4145_ECDH_COFACTOR_DERIVE є алгоритмом, що містить обчислення спільного секрету ZZ з кофакторним множенням, об'єктний ідентифікатор якого такий:

```
id-dhSinglePass-cofactorDH-gost34311kdf-scheme OBJECT
    IDENTIFIER ::= { iso(1) member-body(2) Ukraine(804) root(2)
        security(1) cryptography(1) pki(1) pki-alg(1) pki-alg-asym(3)
        dhSinglePass-cofactorDH-gost34311kdf(4) }
```

Примітка. Відповідно до ДСТУ ISO/IEC 15946-3:2005 кофакторне множення є одним із методів запобігання методам «атаки малої підгрупи» (деталі атаки наведені у міжнародних рекомендаціях RFC 2785), тому рекомендується використовувати механізм CKM_DSTU4145_ECDH_COFACTOR_DERIVE.

Механізми визначаються такими ідентифікаторами:

```
CKM_DSTU4145_ECDH_DERIVE = CKM_VENDOR_DEFINED +
0x00420043;
CKM_DSTU4145_ECDH_COFACTOR_DERIVE =
CKM_VENDOR_DEFINED + 0x00420044;
де
CKM_VENDOR_DEFINED = 0x80000000.
```

Для зазначених механізмів *ulMinKeySize* та *ulMaxKeySize* поля структури CK_MECHANISM_INFO визначають відповідно мінімальне та максимальне число біт степеню основного поля ($163 \leq m \leq 509$), які підтримуються.

Інформація про механізми («CK_MECHANISM_INFO») повинна бути такою:

```
ulMinKeySize    = 163;
ulMaxKeySize    = 509;
Flags           = CKF_DERIVE | CKF_EC_F_2M
                 | CKF_EC_ECPARAMETERS | CKF_EC_NAMEDCURVE
                 | CKF_EC_UNCOMPRESS [| CKF_EC_COMPRESS]
                 [| CKF_HW]
```

У випадку апаратної підтримки у полі Flags повинен стояти додатковий прапорець CKF_HW.

Ключ шифрування ключів КШК формується функцією C_DeriveKey (див. розділ 4.12.5) за допомогою заданого механізму ECDH.

Параметр функції hBaseKey повинен містити дескриптор (handle) особистого ключа відправника (для статичної схеми) або дескриптор (handle) віртуального особистого ключа відправника (для динамічної схеми) у разі функції WRAP, та дескриптор (handle) особистого ключа одержувача у разі функції UNWRAP.

У шаблоні атрибутів КШК, який створюється, повинні бути присутніми обов'язкові атрибути, які зазначені для WRAP/ UNWRAP механізму

Якщо для захисту ключа використовується механізм CKM_GOST28147_WRAP, то шаблон атрибутів ключа КШК за замовчуванням визначено у розділі 5.3.6 цих специфікацій.

У разі використання інших механізмів, наприклад AES, параметри за замовчуванням беруться згідно з стандартом PKCS#11.

Примітка:

- Довжина створеного функцією C_DeriveKey() КШК ключа CKA_VALUE_LEN за замовчуванням дорівнює 32 байти. Якщо створюється ключ іншої довжини (наприклад, AESWrap 128, 192 біт), то цей атрибут необхідно явно задавати.

Дія механізму CKM_DSTU4145_ECDH_COFACTOR_DERIVE повністю аналогічна дії механізму CKM_DSTU4145_ECDH_DERIVE, за винятком використання кофакторного множення (cofactor) безпосередньо в тілі механізму.

Приклад:

```
CK_OBJECT_HANDLE hPublicKey1, hPrivateKey1;  
CK_OBJECT_HANDLE hPublicKey2, hPrivateKey2;
```

```
CK_MECHANISM mech = {CKM_DSTU4145_KEY_PAIR_GEN, NULL, 0};
```

```
rv = C_GenerateKeyPair (hSession, &mech,  
NULL, 0 , NULL, 0, &hPublicKey1, &hPrivateKey1);  
if (rv != CKR_OK) {  
    printf("\nC_GenerateKeyPair failed: %d", rv);  
    return;  
}
```

```
rv = C_GenerateKeyPair (hSession, &mech,  
NULL, 0 , NULL, 0, &hPublicKey2, &hPrivateKey2);  
if (rv != CKR_OK) {  
    printf("\nC_GenerateKeyPair failed: %d", rv);  
    return;  
}
```

```
CK_BYTE      PubValue[256] = {0,};  
CK_ULONG     PubLen = sizeof (PubValue);
```

```
CK_ATTRIBUTE TemplEcPoint[] = {  
    {CKA_EC_POINT, PubValue, PubLen},  
};
```

```

rv = C_GetAttributeValue(hSession, hPubKey2,
                        TemplEcPoint, NUMITEMS (TemplEcPoint));
if (rv) {
    printf("\nC_GetAttributeValue: %d", rv);
    return;
}

CK_DSTU4145_ECDH_DERIVE_PARAMS params = {0,};

CK_BYTE SharedData[64] = {..};

params.ulSharedDataLen = sizeof(SharedData);
memcpy(params.pSharedData, SharedData, sizeof(SharedData));

PubLen = TemplEcPoint->valueLen;
memcpy(params.PublicData, PubValue, PubLen);

params.kdf = CKD_GOST34311_KDF;

CK_OBJECT_HANDLE  hKEK;

mech.mechanism = CKM_DSTU4145_ECDH_DERIVE;

rv = C_DeriveKey(hSession, &mech, hPriKey1,
                TemplSecretKey, NUMITEMS(TemplSecretKey), &hKEK);
if (rv != CKR_OK) {
    printf("\nC_DeriveKey: %d", rv);
    return;
}

```

6. Політики безпеки

6.1. Визначення

Цим розділом встановлюються базові вимоги безпеки (політики безпеки), яким повинні задовольняти криптографічні засоби захисту інформації (надалі криптографічний модуль), що реалізують інтерфейс прикладного програмування Cryptoki відповідно до міжнародного стандарту PKCS#11, та призначені для захисту ключових даних та іншої чутливої інформації/даних (інформації з обмеженим доступом), що вимагає захисту.

Стандарт PKCS#11 призначений для захисту конфіденційних криптографічних ключів навіть при підключенні до небезпечного хоста. Але у той же час API інтерфейсів токена Cryptoki, відповідно до стандарту PKCS#11, як відомо, має певні уразливості, що пов'язані з особливостями реалізації стандарту PKCS#11:

- J. Clulow. On the security of PKCS#11. In 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003), pages 411–425, 2003.,
- S. Delaune, S. Kremer, and G. Steel. Formal analysis of PKCS#11. In Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08), pages 331–344, Pittsburgh, PA, USA, June 2008, IEEE Computer Society Press.

Оскільки різні реалізації криптографічних модулів впроваджують різні підмножини цього стандарту, то ці уразливості можуть бути критичними чи не мати місця взагалі.

Призначення політики безпеки – визначити такі вимоги до реалізації та конфігурування криптографічних модулів, що дозволять створити максимально стійкі до втручання криптографічні модулі безпеки, які відповідають певним вимогам безпеки.

6.1.1. Модель порушника

Модель порушника дотримується підходу, що криптографічний модуль при підключенні до хоста є під повним контролем порушника щодо використання API інтерфейсів токена. Порушник може викликати команду Cryptoki в будь-якому порядку в якому забажає, використовуючи будь-які значення, що він знає.

Не передбачається, що порушник може грубо «зламати» криптографічний алгоритм або вчинити щось подібне, таким чином він може тільки читати криптографічні повідомлення, якщо він має правильний ключ.

Безпека аналізується як досяжність в моделі для станів атак, тобто станів, де порушник знає чи може отримати значення ключа, що зберігається у пристрої з атрибутом чутливості (CKA_SENSITIVE), встановленим у CK_TRUE, або

атрибутом вилучаємості (CKA_EXTRACTABLE) зі значенням CK_FALSE, чи іншими атрибутами, призначеними для забезпечення безпеки зберігання ключа.

Криптографічні пристрої, які підтримують PKCS # 11, мають такі потенційні моделі порушника:

- Зловмисний офіцер безпеки, який зловживає повноваженнями свого становища і своїм доступом до пристрою та функцій управління користувачами;
- Зловмисний користувач, який використовує його авторизований доступ до пристрою;
- Зловмисна третя сторона/ особа, яка отримує доступ до токена через якісь засоби.

По суті, ці порушники можуть отримати доступ до сесії, або отримати доступ до пристрою під час сесії (наприклад, шляхом введення повідомлень у лінії зв'язку) або мають знання пароля (ПІН-коду доступу).

6.1.2. Моделі атак

Стійкі до втручання криптографічні токени, такі як смарт-картки та ключі USB є найбільш загальними компонентами розподілених систем, що розгортаються, як правило, в не занадто безпечних або навіть небезпечних умовах. Вони призначаються, наприклад, для виконання автентифікації, захисту криптографічних значень (ключів тощо) від шкідливих програм і для полегшення безпечного входу для різних застосувань, починаючи з онлайн-банкінгу.

Токен повинен надавати інтерфейс Cryptoki для зовнішнього оточення (прикладних програм застосування), що призначений для виконання криптографічних функцій і операцій управління ключами. Cryptoki є критичним: він повинен бути розроблений таким чином, що навіть якщо пристрій буде знаходитися в контакті з зловмисними програмами, які є загрозою для хост-комп'ютера, то криптографічні значення повинні зберігатися в таємниці.

Відповідно до стандарту PKCS#11, застосування ініціюють сесії з криптографічними токенами з підтримкою PIN-коду. Якщо шкідливий код запущено на хост-комп'ютері, то PIN-код користувача може легко бути перехоплений, наприклад, логером клавіатури або за допомогою підробленого драйвера пристрою, що дозволяють зловмиснику створити його власну сесію з пристроєм.

Відповідно до стандарту PKCS#11, після ініціювання сесії, застосування може отримати чрез Cryptoki доступ до об'єктів, які зберігаються у токені, наприклад, ключів.

Cryptoki звертається до об'єктів через дескриптори об'єктів. Загалом, значення дескриптора, наприклад, для секретного ключа, не розкриває будь-якої інформації про фактичне значення ключа. Об'єкти мають атрибути, що визначають певні властивості об'єкта. Наприклад, чи ключ може використовуватися для шифрування даних, чи для шифрування інших ключів.

Нові об'єкти можуть бути створені функцією створення або копіювання об'єктів, генерації ключа або при «розгорненні» зашифрованого ключа. В усіх випадках повертаються значення нових дескрипторів.

Коли функція Cryptoki викликається з посиланням на певний об'єкт, то токен спочатку повинен перевірити, що атрибути об'єкта дозволяють йому бути використаним для цієї функції. Наприклад, якщо викликається функція шифрування з дескриптором ключа, то ключу необхідно мати набір атрибутів шифрування (CKA_ENCRYPT, CKA_DECRYPT).

Для того, щоб захистити ключ від розкриття, його атрибут чутливості (CKA_SENSITIVE) повинен бути встановлений у CK_TRUE. Це означає, що запит на перегляд значення ключа об'єкта призведе до повідомлення помилки. Відповідно до стандарту PKCS#11, після того, як було встановлено значення атрибуту чутливості у CK_TRUE, воно не може бути встановлено у CK_FALSE. Це дає основні власності безпеки, заявлені у стандарті: атаки, навіть якщо вони викликали шкоду для хост-машини при одержанні PIN-коду, не можуть скомпрометувати ключі з позначкою «чутливі», оскільки чутливий ключ завжди буде залишатися чутливим.

Такий ключ може експортуватися за межі криптографічного модулю лише, якщо він зашифрований за допомогою іншого ключа, і тільки тоді, коли його атрибут вилучення (CKA_EXTRACTABLE) встановлено у значення CK_TRUE. Об'єкт з атрибутом вилучення, встановленим у CK_FALSE, не може експортуватися за межі модулю, і крім того, значення CK_FALSE не може бути встановлено у CK_TRUE. Захист ключів, по суті, спирається на атрибути чутливості та вилучення.

6.1.2.1. Атака на розмежування ключів

Атака на розмежування ключів (Key Separation Attack) – це атака, де встановлюються атрибути ключа таким чином, щоб надати ключам суперечливі ролі через встановлення їх атрибутів, використовуючи функції C_SetAttributeValue, чи C_CopyObject, чи інш.

Наприклад, суперечливі ролі має ключ з атрибутами, встановленими для розшифрування зашифрованого тексту/ даних та для «обгорнення»/ зашифрування ключів. Для того, щоб визначити значення чутливого ключа, зломисник просто обгортає/ зашифровує його на відомому йому ключі КШК (ключі шифрування ключів), вилучає із токена, а потім розшифровує.

Відповідно до форматів обгорнення, визначених стандартом PKCS#11, криптографічний модуль не може визначити чи є довільні рядки бітів криптографічним ключем, чи вони є деякою частиною звичайного (незашифрованого) тексту.

Таким чином, коли виконується команда розшифрування, модуль не має можливості визначити, що пакет, що дешифрується, містить ключ.

Для попередження атаки на розмежування ключів, ключі повинні бути розділені щодо їх застосування шляхом виконання таких правил (політик атрибутів):

1. Конфліктуючі атрибути (Conflicting attributes): Ключі обгорнення (СКА_WRAP, СКА_UNWRAP) не повинні застосовуватися для шифрування даних (СКА_ENCRYPT, СКА_DECRYPT), і навпаки.
2. Пов'язані атрибути (Associated attributes): Особисті ключі підпису (СКА_SIGN) повинні мати атрибут чутливості (СКА_SENSITIVE), встановлений у СК_TRUE, та пов'язаний атрибут вилучення (СКА_EXTRACTABLE), встановлений у значення СК_FALSE. Допускається встановлення атрибуту вилучення (СКА_EXTRACTABLE) у значення СК_TRUE лише безпосередньо у шаблоні генерації ключової пари з метою негайного створення резервної копії ключа; після створення резервної копії атрибут вилучення (СКА_EXTRACTABLE) повинен бути встановлений у значення СК_FALSE.
3. Прикріплені атрибути чутливості (Sticky attributes of sensitivity): Особисті ключі підпису (СКА_SIGN), ключі обгорнення (СКА_WRAP, СКА_UNWRAP) та ключі шифрування (СКА_ENCRYPT, СКА_DECRYPT) повинні мати атрибут чутливості (СКА_SENSITIVE), встановлений у СК_TRUE.
4. Прикріплені атрибути вилучаємості (Sticky attributes of extraction): Особисті ключі підпису (СКА_SIGN), ключі обгорнення (СКА_WRAP, СКА_UNWRAP) та ключі шифрування даних (СКА_ENCRYPT, СКА_DECRYPT) повинні мати атрибут вилучення (СКА_EXTRACTABLE), встановлений у значення СК_FALSE.
5. Прикріплені атрибути модифікуємості (Sticky attributes of modifiability): Усі секретні та особисті ключі повинні мати атрибут СКА_MODIFIABLE, встановлений у СК_FALSE, чим забороняється змінювати (модифікувати) як значення ключа, так і будь які його атрибути.

Зазначені політики атрибутів повинні перевірятися під час виконання відповідних функцій Cryptoki, які використовують ключі, та у разі невідповідності повертати помилку CKR_TEMPLATE_INCONSISTENT.

Примітка. Атрибут СКА_LOCAL також може використовуватися, як додаткова властивість безпеки, тобто, що ключ створено функцією генерування ключів (C_GenerateKey чи C_GenerateKeyPair). Однак відсутність атрибуту СКА_LOCAL = СК_TRUE не означає небезпечність. Дісно, ключі можуть бути безпечно відновлені з резервної копії, чи безпечно вироблені функцією C_DeriveKey, але мають при цьому, відповідно до стандарту PKCS#11, атрибут СКА_LOCAL = СК_FALSE. Для імпортованих ключів також існує можливість зберігати публічні ключі з ознакою довіри у атрибуті СКА_TRUSTED. Атрибут СКА_TRUSTED може бути встановлено у СК_TRUE тільки операцією Офіцера безпеки (SO), який повинен авторизуватися перед встановленням цього атрибуту.

6.1.2.2. Атаки API симетричних ключів

Атаки API симетричних ключів (Symmetric Key API Attacks) стосуються певних механізмів з симетричними ключами, які відповідно до PKCS#11, може підтримувати функція C_DeriveKey. Це такі механізми:

- CKM_CONCATENATE_BASE_AND_KEY,
- CKM_CONCATENATE_BASE_AND_DATA,
- CKM_CONCATENATE_DATA_AND_BASE,
- CKM_XOR_BASE_AND_DATA,
- CKM_EXTRACT_KEY_FROM_KEY.

До зазначених механізмів можуть бути застосовані такі атаки:

- Зменшення простору ключа (Reduced Key Space): Використанням механізму CKM_EXTRACT_KEY_FROM_KEY можна виділити підмножину бітів з заданого ключа для створення коротких ключів. Можуть бути використані для зниження ключового простору, необхідне для пошуку. Наприклад, можна було б отримати з 40-бітного ключа DES для створення 40-бітних ключів RC2, які потім можуть бути знайдені вичерпними методами. Фактичний простір може бути менше через існування бітів парності в ключах DES. Це потенційно небезпечний механізм. Механізм CKM_XOR_BASE_AND_DATA також забезпечує простий метод, який може бути використано для зниження ключового простору, необхідного для пошуку, генеруючи велику кількість (відомих) пов'язаних ключів.

- Паралельний пошук (Parallel Search): Механізм CKM_XOR_BASE_AND_DATA забезпечує простий метод для обмеження (exclusive) чи отримання відомих шаблонів (patterns) від ключа. Це може бути використано для зниження ключового простору, необхідний для пошуку, генеруючи велику кількість (відомих) пов'язаних ключів.

- Атаки пов'язаних ключів (Related Key Attack): Використанням механізму CKM_XOR_BASE_AND_DATA, можна створити набір пов'язаних ключів, з якою для виконання відповідних основних атак. Наприклад, це може бути використано для зниження 3-ключового 3DES до стану, який тільки трохи сильніше, ніж простий DES (зниження ключового простору пошуку до 256 операцій, щоб ізолювати ключовий компонент).

Для попередження атаки API симетричних ключів, зазначені вище механізми (умовно «Чистий PKCS11») повинні бути заборонені для використання (відключені).

6.1.2.3. Атаки API відкритих ключів

Атаки API відкритих ключів (Public Key API Attacks) стосуються асиметричних ключів. До них належать, зокрема, такі атаки:

- Троянський відкритий ключ (Trojan Public Key): Відкриті ключі в PKCS#11 API є відкритими на токені без будь-яких додаткових перевірок достовірності. При цьому можна використовувати будь-який відкритий ключ для функції C_WrapKey. Це дозволяє зломисникові використовувати відкритий «троянський» ключ, за яким він знає відповідний закритий ключу (як правило, зломисник, швидше за все, буде генерувати пару ключів сам). Він запитує PKCS#11 токен виконати експорт цільового ключа (об'єкт атаки), використовуючи його наданий відкритий ключ. Оскільки зломисник знає відповідний особистий ключ (дескриптор), то він може легко відновити цільовий ключ. Цей простий метод може бути використаний для відновлення всіх ключів, що можуть експортуватися, незалежно від того, чи є вони симетричні, чи особисті асиметричні ключі. Таким чином, ясно, що відкритий ключ повинен пройти перевірку справжності перед його використанням, щоб переконатися, що він дійсно має право на експорт даного ключа.

- Троянський ключ обгорнення (Trojan Wrapped Key): Як і у випадку неаутентифікованого використання відкритих ключів, не існує методу, щоб переконатися, що ключ обгорнення токена є дійсно автентичним/ не підробленим. Атакуючий вибирає довільний ключ K, який він потім «обгортає» відомим відкритим ключем. Потім він викликає C_UnwrapKey функцію для цього «троянсько» загорнутого ключа і імпортує ключ як новий ключ в систему. Таким чином, існує вимога про надання засобів перевірки справжності та походження ключа загортання.

Для попередження атаки API відкритих ключів, ключі повинні мати додаткові атрибути, що гарантують їх надійність (безпечність), та шляхом виконання таких правил (політик атрибутів):

1. Локальні об'єкти (Local Objects): Об'єкти і тільки такі об'єкти, що створюються механізмами генерації на токені, отримують атрибут SKA_LOCAL, встановлений у CK_TRUE.
2. Довірені об'єкти (Trusted Objects): Об'єкти користувача отримують атрибут SKA_TRUSTED, встановлений у CK_TRUE, виключно операцією Офіцера безпеки (SO). Застосовується, зокрема, для об'єкта, відновленого безпечним способом із резервної копії. Користувач може відновити об'єкт із резервної копії та встановити йому атрибути SKA_SENSITIVE, SKA_EXTRACTABLE та інші. Але лише SO може встановити атрибут SKA_TRUSTED у CK_TRUE. Примітка. Може застосовуватися до відкритих об'єктів, чи таких, доступ до яких може мати SO. У разі відновлення ключової пари, атрибут SKA_TRUSTED у CK_TRUE встановлюється для відкритого ключа, після чого його відповідний особистий ключ вважається також довіреним (неявно, через свій відкритий ключ має встановлений атрибут SKA_TRUSTED).
3. Сесійний ключ обгорнення (Session Wrap Key): Механізми виробництва ключа (функція C_DeriveKey), що використовують «Локальні об'єкти» (SKA_LOCAL = CK_TRUE) чи «Довірені об'єкти» (SKA_TRUSTED = CK_TRUE), створюють ключі обгорнення з атрибутом SKA_TOKEN,

встановлений у CK_FALSE. Сесійний ключ автоматично знищується після закриття сесії та не може бути зчитаний іншими застосуваннями (іншими сесіями). Виконати функцію обгорнення локального чи довіреного об'єкта з атрибутом SKA_EXTRACTABLE, встановленими у CK_TRUE (наприклад, сесійного ключа шифрування даних чи особистого ключа, згенерованого в токені чи відновленого безпечним способом із резервної копії), можна виключно, якщо ключ обгорнення є сесійним ключем з атрибутом SKA_EXTRACTABLE, встановленими у CK_FALSE.

6.1.3. Загальні політики

Цими специфікаціями передбачається два якісних рівні/ політики безпеки (від нижчого до вищого рівня) криптографічних модулів:

- PKCS #11сумісна політика;
- Рекомендована загальна політика;
- Безпечна загальна політика.

PKCS#11сумісна політика (нижчий рівень безпеки) призначається для цілей тестування криптографічного модулю та правильності реалізації його PKCS#11 механізмів.

Рекомендована загальна політика (більш високий рівень безпеки) призначається для цілей забезпечення безпеки експлуатації криптографічного модулю, що функціонує у довіреному/ захищеному середовищі, де мало ймовірно є перехоплення даних під час обміну між криптографічним модулем та його PKCS#11 інтерфейсом Cryptoki. Наприклад, модуль - внутрішній PC адаптер PCI-E; PC хост-комп'ютер є захищеним від перехоплення інформації, що циркулює між інтерфейсом Cryptoki адаптером PCI-E мккриптографічного модулю.

Безпечна загальна політика (найбільш високий рівень безпеки) призначається для цілей забезпечення безпеки експлуатації криптографічного модулю для захисту інформації з обмеженим доступом. Ця політика дозволяє криптографічному модулю за допомогою відповідних процедурних засобів забезпечувати принаймні, що:

- а) забезпечується секретність ключових даних, які використовуються для вироблення підпису;
- б) ключові дані, які використовуються для вироблення підпису, із значною долею впевненості не можуть вилучатися з цих модулів, а підпис захищається від підробки за допомогою використання доступних технологій;
- в) ключові дані, які використовуються для вироблення підпису, можуть бути надійно захищені законною особою (підписувачем) від використання їх іншими особами;

г) цілісність даних/ повідомлень під час взаємодії прикладної програми підписувача з модулем гарантується.

Криптографічний модуль, у якого встановлена «Безпечна загальна політика» будемо називати «Безпечний криптографічний модуль».

Експлуатаційна документація криптографічного модулю повинна визначити реалізовану політику безпеки модулю. Політика безпеки повинна включати правила, що зазначні у цьому розділі, і правила, отримані в результаті будь-яких додаткових вимог, встановлених виробником.

6.2. Опції безпеки

Для реалізації політики безпеки криптографічний модуль може мати ряд опцій / прапорів безпеки для детального вибору/ встановлення вибраної політики. Сукупністю встановлених опцій і визначається конкретна політика безпеки.

Опції безпеки позначаються таким чином:

Таблиця 6.1. Опції безпеки

№ п/п	Назва	Позначення
1.	Чистий PKCS11 (Pure PKCS11)	PP
2.	Захист крипто операцій (No Public Crypto)	NPC
3.	Збільшений рівень безпеки (Increased Security Level)	ISL
4.	Захищений ПІН (No Clear PIN)	NCP
5.	Захист аутентифікацією (Auth Protection)	AP
6.	Підписування обміну (Secure Messaging Signing)	SMS
7.	Шифрування обміну (Secure Messaging Encryption)	SME
8.	Зіпсувати перед оновленням (Tamper Before Upgrade)	TBU
9.	Дозволити доменні параметри користувача (User Specified Domain Parameters Allowed)	USDPA
10.	Режим заблоковано (Mode Locked)	ML

Далі надається детальний опис опцій політики безпеки.

6.2.1. Чистий PKCS11

Опція «Чистий PKCS11» (Pure PKCS11, PP) індикує про те, що наступні механізми будуть поводитись, як вимагає стандарт PKCS#11 v2.20:

CKM_CONCATENATE_BASE_AND_KEY
CKM_CONCATENATE_BASE_AND_DATA
CKM_CONCATENATE_DATA_AND_BASE
CKM_EXTRACT_KEY_FROM_KEY
CKM_XOR_BASE_AND_DATA

Увага!

Встановлення цієї опції піддає ризику безпеку. Досвідчений злоумисник має можливість використати вразливість цих механізмів у той час, коли цей флаг встановлено. Для забезпечення безпеки ця опція повинна бути виключена.

6.2.2. Захист крипто операцій

Опція «Захист крипто операцій» (No Public Crypto, NPC) індикує про те, що користувач не може виконувати криптографічні операції без первинної аутентифікації.

Кожен токен, у системі повинен мати встановлену опцію/ прапор CKF_LOGIN_REQUIRED (див. розділ 2.7.2) для надання інформації інтерфейсу застосування про необхідність аутентифікації користувача до виконання будь-яких крипто операцій.

6.2.3. Збільшений рівень безпеки

Включення опції «Збільшений рівень безпеки» (Increased Security Level, ISL) означає наступне:

- Механізм CKM_EXTRACT_KEY_FROM_KEY (див. стандарт PKCS#11) відключено;
- Усі секретні та особисті ключі повинні мати атрибут SCA_MODIFIABLE, встановлений у CK_FALSE, чим забороняється змінювати (модифікувати) як значення ключа, так і будь які його атрибути;
- Зміна атрибута SCA_MODIFIABLE із CK_FALSE до CK_TRUE, використовуючи функцію C_CopyObject, не дозволяється.
- Не дозволяється використання особистих ключів з атрибутом SCA_EXTRACTABLE встановленим у CK_TRUE.
- Якщо використовується функція C_DeriveKey, то SCA_EXTRACTABLE атрибут повинен бути встановлений у CK_FALSE (ключ шифрування ключів ніколи не залишає криптографічний модуль). Інакше повертається помилка CKR_TEMPLATE_INCONSISTENT.
- Якщо використовується функція C_UnwrapKey, то SCA_EXTRACTABLE та SCA_SENSITIVE атрибути нового (розшифрованого) ключа повинні бути встановлені у CK_TRUE, (ключ шифрування даних може бути експортовано лише у захищеному вигляді). Інакше повертається помилка CKR_TEMPLATE_INCONSISTENT.
- Створений функцією C_UnwrapKey (розшифрований) ключ або створений функцією C_DeriveKey ключ повинен бути сесійним, тобто мати атрибут CK_TOKEN, встановлений у значення CK_FALSE. Якщо

задано відповідний шаблон ключа, що містить інше значення атрибуту CK_TOKEN, то повинна повертатися помилка CKR_TEMPLATE_INCONSISTENT.

- Ніяка функція, що використовує об'єкти із встановленим у значення CK_TRUE атрибутом SCA_SENSITIVE, не повинна виконуватися, якщо ці об'єкти мають атрибут SCA_PRIVATE, не встановлений у CK_TRUE. У цьому випадку повинна повертатися помилка CKR_TEMPLATE_INCONSISTENT;
- Забезпечення політики «Довірений ключ обгорнення» (Trusted Wrap Key): Механізми виробництва ключа (функція C_DeriveKey), що використовують «Локальні об'єкти» (SCA_LOCAL = CK_TRUE) чи «Довірені об'єкти» (SCA_TRUSTED = CK_TRUE), повинні створювати ключі обгорнення з атрибутом SCA_TRUSTED, встановленим у CK_TRUE. Виконати функцію обгорнення локального чи довіреного об'єкта з атрибутами SCA_SENSITIVE, SCA_EXTRACTABLE, встановленими у CK_TRUE (наприклад, сесійного ключа шифрування даних чи особистого ключа, згенерованого в токені чи відновленого безпечним способом із резервної копії), можна виключно, якщо ключ обгорнення є довіреним.

6.2.4. Захищений ПІН

Опція «Захищений ПІН» (No Clear PIN, NCP) визначає, що коли вона встановлена, то відбувається наступне:

- ПІН користувача (User PIN) або інша чутлива інформація не може передаватися у відкритому вигляді;
- Вмикається безпечне шифрування повідомлень для запитів до модуля, що містять критичну або чутливу інформацію у цих запитах;
- Вимикаються ті функції, що можуть повертати результат виконання у відкритій формі;
- Генерування ключів з атрибутом SCA_SENSITIVE, встановленим у CK_FALSE, забороняється.

6.2.5. Захист аутентифікацією

Опція «Захист аутентифікацією» (Auth Protection, AP) визначає безпеку обміну даними, гарантуючи цілісність даних під час їх передавання/ приймання. Якщо опція встановлена, то усі повідомлення між криптографічним модулем та інтерфейсом PKCS#11 (Cryptography) використовують додаткову аутентифікацію повідомлень, тобто до кожного повідомлення додається код автентифікації

повідомлення (MAC), що перевіряється стороною приймання. Для формування та перевірки MAC використовується сеансовий ключ, що виробляється за стандартизованим алгоритмом (наприклад, Diffie-Hellman Key Exchange/ Diffie-Hellman Key Agreement чи іншим), який підтримується модулем.

6.2.6. Підписування обміну

Опція «Підписування обміну» (Secure Messaging Signing, SMS) індикує про те, що для кожного повідомлення у обох напрямках обміну між інтерфейсом PKCS#11 (Cryptoki) та модулем застосовується безпечна аутентифікація повідомлень. Цілісність кожного повідомлення забезпечується його підписанням у цифровій формі (кодом аутентифікації повідомлень, MAC) та відповідною перевіркою на іншій стороні.

Ця опція подібна до опції «Захист аутентифікацією» (Auth Protection), але включає аутентифікацію кожного повідомлення у обох напрямках під час передавання/ приймання. Для формування та перевірки MAC використовується сеансовий ключ, що виробляється за стандартизованим алгоритмом (наприклад, Diffie-Hellman Key Exchange/ Diffie-Hellman Key Agreement чи іншим), який підтримується модулем.

6.2.7. Шифрування обміну

Опція «Шифрування обміну» (Secure Messaging Encryption, SME) визначає встановлення шифрованого каналу між інтерфейсом PKCS#11 (Cryptoki) та модулем. Опція індикує наступне:

- PIN-код користувача або інша чутлива інформація (CKA_SENSITIVE = CK_TRUE) не може бути передана між Cryptoki та модулем в не зашифрованому вигляді;
- Коли безпечний обмін повідомленнями увімкнений, то це зобов'язує шифрувати кожне повідомлення у обох напрямках між Cryptoki та модулем;
- Відключені ті функції, що повертають відкритий (не зашифрований) результат чутливих даних (CKA_SENSITIVE = CK_TRUE);
- Не дозволяється генерування ключів з атрибутом CKA_SENSITIVE, встановленим у CK_FALSE.

Ця опція подібна до «Захищений ПІН» (No Clear PIN), але включає шифрування кожного повідомлення у обох напрямках обміну. Для шифрування використовується сеансовий ключ, що виробляється за стандартизованим алгоритмом (наприклад, Diffie-Hellman Key Exchange/ Diffie-Hellman Key Agreement чи іншим), який підтримується модулем.

6.2.8. Зіпсувати перед оновленням

Програмне забезпечення модулю для події «Спроба несанкціонованного доступу» повинно забезпечувати виконання операції стирання внутрішньої захищеної пам'яті.

Опція «Зіпсувати перед оновленням» (Tamper Before Upgrade, TBU) індикує про те, що програмне забезпечення події «Спроба несанкціонованного доступу» повинно також виконуватись в будь якому із таких випадків:

- Оновлення програмного забезпечення модулю;
- Завантаження функціонального модулю/ аплету (для модулів, що підтримують таке завантаження та/чи підтримують мульти-застосування – multy-application);
- Відключення функціонального модулю/ аплету (для модулів, що підтримують таке завантаження та/чи підтримують мульти-застосування – multy-application).

6.2.9. Дозволити доменні параметри користувача

Опція «Дозволити доменні параметри користувача» (User Specified DomainParameters Allowed, USDPA) визначає таку можливість:

- Використовувати створені користувачем об'єкти SBOX (див. розділ 5.2), що не є вмонтованими (native) у модуль об'єктами. Якщо опцію не встановлено, то неможливо виконати функції із застосуванням створеного користувачем об'єкту SBOX, наприклад, виконати функцію генерування ключа C_GenerateKey та інші функції усіх механізмів, де використовується цей створений об'єкт SBOX. Ці функції повертають код помилки SKR_OPERATION_NOT_PERMITTED
- Використовувати створені користувачем доменні параметри еліптичних кривих, об'єкти EC_PARAMS (див. 5.5.2), що не є вмонтованими (native) у модуль об'єктами. Якщо цю опцію не встановлено, то неможливо виконати функції із використанням створеного користувачем об'єкту EC_PARAMS, наприклад, генерування ключової пари C_GenerateKeyPair та інші функції усіх механізмів, де використовується цей створений об'єкт. Ці функції повертають код помилки SKR_OPERATION_NOT_PERMITTED.

6.2.10. Режим заблоковано

Опція «Режим заблоковано» (Mode Locked, ML) означає блокування змін режиму безпеки. Якщо опцію встановлено, то ні ця, і ні яка інша опція безпеки не

може бути змінена. Нова політика безпеки може встановлюватися тільки після ініціювання події «спроба несанкціонованого доступу».

6.3. Рекомендовані політики безпеки

Рекомендовані політики безпеки та їх встановлені опції наведені у таблиці:

Таблиця 6.2. Рекомендовані політики безпеки

Політики безпеки	Вплив флагів безпеки на політики безпеки									
	Чистий PKCS11	Захист крипто операції	Збільшений рівень безпеки	Захищений ПН	Захист аутентифікацією	Підписування обміну	Шифрування обміну	Зіпсування перед оновленням	Дозволити доменні параметри користувача	Режим заблоковано
PKCS#11 сумісна політика	+	*	*	*	*	*	*	*	*	*
Загальна політика безпеки	-	+	+	+	-	-	-	+	*	+
Максимальна політика безпеки	-	+	+	+	+	+	+	+	*	+

Позначення:

- опція обов'язково відсутня для політики;
- + опція обов'язково присутня для політики;
- * опція може бути присутня чи відсутня.

Додаток А. Початкові коди файлу заголовків

1. Константи, що визначаються у даному документі.

```
// key types
#define CKK_GOST28147 (CKK_VENDOR_DEFINED + 0x00420111)
#define CKK_DSTU4145 (CKK_VENDOR_DEFINED + 0x00420131)

// mech
#define CKM_GOST28147_ECB (CKM_VENDOR_DEFINED + 0x00420011)
#define CKM_GOST28147_OFB (CKM_VENDOR_DEFINED + 0x00420012)
#define CKM_GOST28147_CFB (CKM_VENDOR_DEFINED + 0x00420013)
#define CKM_GOST28147_MAC (CKM_VENDOR_DEFINED + 0x00420014)
#define CKM_GOST28147_KEY_WRAP (CKM_VENDOR_DEFINED + 0x00420015)
#define CKM_GOST34311 (CKM_VENDOR_DEFINED + 0x00420021)
#define CKM_DSTU4145 (CKM_VENDOR_DEFINED + 0x00420031)
#define CKM_DSTU4145_WITH_GOST34311 (CKM_VENDOR_DEFINED + 0x00420032)
#define CKM_GOST28147_KEY_GEN (CKM_VENDOR_DEFINED + 0x00420041)
#define CKM_DSTU4145_KEY_PAIR_GEN (CKM_VENDOR_DEFINED + 0x00420042)
#define CKM_DSTU4145_ECDH_DERIVE (CKM_VENDOR_DEFINED + 0x00420043)
#define CKM_DSTU4145_ECDH_COFACTOR_DERIVE (CKM_VENDOR_DEFINED + 0x00420044)

// key derive func
#define CKD_GOST34311_KDF (CKD_VENDOR_DEFINED + 0x00420211)

// attrib
#define CKA_SBOX (CKA_VENDOR_DEFINED + 0x00420311)

// errors
#define CKR_SBOX_NOT_FOUND (CKR_VENDOR_DEFINED + 0x00420403)
#define CKR_EC_PARAMS_NOT_FOUND (CKR_VENDOR_DEFINED + 0x00420406)
#define CKR_EC_PARAMS_INVALID (CKR_VENDOR_DEFINED + 0x00420409)
#define CKR_PRIVATE_KEY_NOT_FOUND (CKR_VENDOR_DEFINED + 0x00420404)
#define CKR_PUBLIC_KEY_NOT_FOUND (CKR_VENDOR_DEFINED + 0x00420405)
#define CKR_EC_POINT_INVALID (CKR_VENDOR_DEFINED + 0x00420414)
#define CKR_EC_KEY_INVALID (CKR_VENDOR_DEFINED + 0x00420413)
#define CKR_ID_ALREADY_EXIST (CKR_VENDOR_DEFINED + 0x00420416)
#define CKR_OID_INCORRECT (CKR_VENDOR_DEFINED + 0x00420418)
#define CKR_DIAGNOSTIC_ERROR (CKR_VENDOR_DEFINED + 0x00420419)
```

2. Типи даних, що визначаються у даному документі.

```
typedef struct CK_SEED_PARAMS {
    CK_BYTE seed[64];
} CK_SEED_PARAMS;
typedef CK_SEED_PARAMS * CK_SEED_PARAMS_PTR;

typedef struct CK_GOST28147_PARAMS {
    CK_BYTE iv8[8];
} CK_GOST28147_PARAMS;
typedef CK_GOST28147_PARAMS * CK_GOST28147_PARAMS_PTR;

typedef struct CK_GOST34311_PARAMS {
    CK_BYTE sbox[66];
    CK_BYTE iv32[32];
}
```

```

    } CK_GOST34311_PARAMS ;
typedef CK_GOST34311_PARAMS * CK_GOST34311_PARAMS_PTR;

typedef struct CK_DSTU4145_ECDH_DERIVE_PARAMS {
    CK_EC_KDF_TYPE kdf;
    CK_BYTE         SharedData[64];
    CK_ULONG        ulSharedDataLen;
    CK_BYTE         PublicData[128];
} CK_DSTU4145_ECDH_DERIVE_PARAMS;
typedef CK_DSTU4145_ECDH_DERIVE_PARAMS*
        CK_DSTU4145_ECDH_DERIVE_PARAMS_PTR;

```

3. Об'єктні ідентифікатори, що використовуються у даному документі.

OID_AES128_KEY_WRAP	= 2.16.840.1.101.3.4.1.5
OID_AES192_KEY_WRAP	= 2.16.840.1.101.3.4.1.25
OID_AES256_KEY_WRAP	= 2.16.840.1.101.3.4.1.45
OID_GOST28147_KEY_WRAP	= 1.2.804.2.1.1.1.1.1.5
OID_DSTU4145_POLY_CURVE_163	= 1.2.804.2.1.1.1.1.3.1.1.2.0
OID_DSTU4145_POLY_CURVE_167	= 1.2.804.2.1.1.1.1.3.1.1.2.1
OID_DSTU4145_POLY_CURVE_173	= 1.2.804.2.1.1.1.1.3.1.1.2.2
OID_DSTU4145_POLY_CURVE_179	= 1.2.804.2.1.1.1.1.3.1.1.2.3
OID_DSTU4145_POLY_CURVE_191	= 1.2.804.2.1.1.1.1.3.1.1.2.4
OID_DSTU4145_POLY_CURVE_233	= 1.2.804.2.1.1.1.1.3.1.1.2.5
OID_DSTU4145_POLY_CURVE_257	= 1.2.804.2.1.1.1.1.3.1.1.2.6
OID_DSTU4145_POLY_CURVE_307	= 1.2.804.2.1.1.1.1.3.1.1.2.7
OID_DSTU4145_POLY_CURVE_367	= 1.2.804.2.1.1.1.1.3.1.1.2.8
OID_DSTU4145_POLY_CURVE_431	= 1.2.804.2.1.1.1.1.3.1.1.2.9
OID_GOST28147_SBOX_1	= 1.2.804.2.1.1.1.1.1.1.10.1
OID_GOST28147_SBOX_2	= 1.2.804.2.1.1.1.1.1.1.10.2
OID_GOST28147_SBOX_3	= 1.2.804.2.1.1.1.1.1.1.10.3
OID_GOST28147_SBOX_4	= 1.2.804.2.1.1.1.1.1.1.10.4
OID_GOST28147_SBOX_5	= 1.2.804.2.1.1.1.1.1.1.10.5
OID_GOST28147_SBOX_6	= 1.2.804.2.1.1.1.1.1.1.10.6
OID_GOST28147_SBOX_7	= 1.2.804.2.1.1.1.1.1.1.10.7
OID_GOST28147_SBOX_8	= 1.2.804.2.1.1.1.1.1.1.10.8
OID_GOST28147_SBOX_9	= 1.2.804.2.1.1.1.1.1.1.10.9
OID_GOST28147_SBOX_10	= 1.2.804.2.1.1.1.1.1.1.10.10

ЗМІСТ

1.	Загальні положення.....	1
2.	Загальна модель та типи даних	3
2.1.	Загальна модель Cryptoki	3
2.2.	Логічна модель токену	4
2.3.	Користувачі	5
2.4.	Символи та аббревіатури.....	6
2.5.	Програми/Застосування та їх використання Cryptoki.....	8
2.5.1.	Програми та процеси	8
2.5.2.	Програми та потоки	9
2.6.	Сесії	10
2.6.1.	Сесія «Тільки для читання».....	10
2.6.2.	Сесія «Читання / запису».....	11
2.6.3.	Сесійні події	11
2.6.4.	Дескриптор сесії та дескриптор об'єкту.....	12
2.6.5.	Можливості сесій.....	13
2.7.	Загальні типи даних	13
2.7.1.	Загальна інформація	13
2.7.2.	Типи слоту та токену	15
2.7.3.	Типи сесії.....	21
2.7.4.	Типи об'єктів.....	23
2.7.5.	Типи даних для механізму.....	26
2.7.6.	Типи для функцій.....	28
2.7.7.	Типи, пов'язані з блокуванням	31
3.	Об'єкти.....	35
3.1.	Загальні визначення.....	35
3.2.	Створення, редагування та копіювання об'єктів.....	36
3.2.1.	Створення об'єктів	36
3.2.2.	Модифікація об'єктів	38
3.2.3.	Копіювання об'єктів.....	38
3.3.	Загальні атрибути об'єктів	38
3.4.	Об'єкти апаратних властивостей.....	39
3.4.1.	Огляд	39
3.4.2.	Об'єкт «Годинник».....	40
3.4.3.	Об'єкт «Монотонний лічильник».....	40
3.4.4.	Об'єкти інтерфейсу користувача.....	41
3.5.	Об'єкти зберігання	43
3.6.	Об'єкти «Дані».....	44
3.7.	Об'єкти «Сертифікат»	45
3.7.1.	Загальні атрибути.....	45
3.7.2.	Об'єкти «X.509 сертифікат» відкритого ключа.....	46
3.7.3.	Об'єкти «WTLS сертифікат» відкритого ключа.....	48
3.7.4.	Об'єкти «X.509 сертифікат атрибутів»	50

3.7.5.	Об'єкти «Ключ»	52
3.7.6.	Загальні атрибути.....	52
3.7.7.	Об'єкти «Відкриті ключі».....	53
3.7.8.	Об'єкти «Особисті ключі»	55
3.7.9.	Об'єкти «Секретні ключі»	57
3.8.	Об'єкти «Механізм»	61
4.	Функції	62
4.1.	Загальна інформація	62
4.2.	Повернені функціями значення	63
4.2.1.	Універсальні повернені значення Cryptoki-функцій.....	63
4.2.2.	Повернені значення Cryptoki-функції для функцій, що використовують дескриптори сесії.....	63
4.2.3.	Повернені значення Cryptoki-функції для функцій, що застосовують токен	64
4.2.4.	Інші повернені значення Cryptoki-функцій.....	65
4.2.5.	Про відносні пріоритети помилок Cryptoki.....	70
4.2.6.	Конвенція щодо значень, які повертають функції до буфера змінної довжини	71
4.3.	Функції загального призначення	71
4.3.1.	C_Initialize	72
4.3.2.	C_Finalize.....	73
4.3.3.	C_GetInfo	74
4.3.4.	C_GetFunctionList.....	75
4.4.	Функції управління слотом та токеном	76
4.4.1.	C_GetSlotList	76
4.4.2.	C_GetSlotInfo.....	78
4.4.3.	C_GetTokenInfo	78
4.4.4.	C_WaitForSlotEvent.....	79
4.4.5.	C_GetMechanismList	81
4.4.6.	C_GetMechanismInfo	82
4.4.7.	C_InitToken.....	83
4.4.8.	C_InitPIN	85
4.4.9.	C_SetPIN.....	86
4.5.	Функції управління сесіями	87
4.5.1.	C_OpenSession.....	88
4.5.2.	C_CloseSession	89
4.5.3.	C_CloseAllSessions.....	90
4.5.4.	C_GetSessionInfo	91
4.5.5.	C_GetOperationState	92
4.5.6.	C_SetOperationState.....	94
4.5.7.	C_Login.....	97
4.5.8.	C_Logout.....	98
4.6.	Функції управління об'єктами	99
4.6.1.	C_CreateObject.....	99
4.6.2.	C_CopyObject	102

4.6.3.	C_DestroyObject	103
4.6.4.	C_GetObjectSize	104
4.6.5.	C_GetAttributeValue	105
4.6.6.	C_SetAttributeValue	108
4.6.7.	C_FindObjectsInit.....	109
4.6.8.	C_FindObjects	110
4.6.9.	C_FindObjectsFinal	111
4.7.	Функції зашифрування	111
4.7.1.	C_EncryptInit	112
4.7.2.	C_Encrypt.....	113
4.7.3.	C_EncryptUpdate.....	114
4.7.4.	C_EncryptFinal.....	115
4.8.	Функції розшифрування	117
4.8.1.	C_Decrypt.....	118
4.8.2.	C_DecryptUpdate	119
4.8.3.	C_DecryptFinal.....	120
4.9.	Функції хешування повідомлення	122
4.9.1.	C_DigestInit.....	122
4.9.2.	C_DigestUpdate.....	124
4.9.3.	C_DigestFinal.....	125
4.10.	Функції обчислення підпису та MAC.....	126
4.10.1.	C_SignInit.....	126
4.10.2.	C_Sign	127
4.10.3.	C_SignUpdate.....	128
4.10.4.	C_SignFinal.....	129
4.11.	Функції для перевіряння підпису та MAC	130
4.11.1.	C_VerifyInit.....	130
4.11.2.	C_Verify.....	131
4.11.3.	C_VerifyUpdate.....	132
4.11.4.	C_VerifyFinal.....	133
4.12.	Функції керування ключами	134
4.12.1.	C_GenerateKey.....	134
4.12.2.	C_GenerateKeyPair	136
4.12.3.	C_WrapKey.....	138
4.12.4.	C_UnwrapKey	140
4.12.5.	C_DeriveKey	142
4.13.	Функції генерування випадкових чисел.....	144
4.13.1.	C_SeedRandom	144
4.13.2.	C_GenerateRandom.....	145
5.	Механізми.....	146
5.1.	Загальні визначення.....	146
5.2.	Об'єкт SBOX.....	147
5.3.	ДСТУ ГОСТ 28147:2009.....	150
5.3.1.	Визначення	150
5.3.2.	Ключові об'єкти.....	150

5.3.3.	Генерація ключів.....	154
5.3.4.	Механізми шифрування.....	156
5.3.5.	Механізм імітовставки.....	158
5.3.6.	Механізм WRAP	160
5.4.	ГОСТ 34.311-95.....	165
5.4.1.	Визначення	165
5.4.2.	Опис механізму	165
5.5.	ДСТУ 4145-2002	167
5.5.1.	Визначення	167
5.5.2.	Об'єкт EC_PARAMS	167
5.5.3.	Ключові об'єкти.....	172
5.5.4.	Прапори інформації про механізми з еліптичними кривими ...	176
5.5.5.	Механізми генерації ключів	177
5.5.6.	Механізми ДСТУ 4145-2002 без хешування	180
5.5.7.	Механізм ДСТУ 4145-2002 з хешуванням	183
5.6.	ECDH Діффі-Хеллмана механізм	185
5.6.1.	Визначення	185
5.6.2.	Параметри механізмів ECDH	185
5.6.3.	Тип KDF-функцій	186
5.6.4.	Алгоритм KDF	187
5.6.5.	Обчисленого спільного секрету ZZ	189
5.6.6.	Механізми ECDH.....	190
6.	Політики безпеки	194
6.1.	Визначення.....	194
6.1.1.	Модель порушника	194
6.1.2.	Моделі атак	195
6.1.3.	Загальні політики	200
6.2.	Опції безпеки	201
6.2.1.	Чистий PKCS11	201
6.2.2.	Захист крипто операцій	202
6.2.3.	Збільшений рівень безпеки.....	202
6.2.4.	Захищений ПІН	203
6.2.5.	Захист аутентифікацією.....	203
6.2.6.	Підписування обміну	204
6.2.7.	Шифрування обміну	204
6.2.8.	Зіпсувати перед оновленням	205
6.2.9.	Дозволити доменні параметри користувача.....	205
6.2.10.	Режим заблоковано	205
6.3.	Рекомендовані політики безпеки	206
	Додаток А. Початкові коди файлу заголовків	207