ГОСУДАРСТВЕННЫЙ СТАНДАРТ РЕСПУБЛИКИ БЕЛАРУСЬ

СТБ 34.101.21-2009

Информационные технологии

ИНТЕРФЕЙС ОБМЕНА ИНФОРМАЦИЕЙ С АППАРАТНО-ПРОГРАММНЫМ НОСИТЕЛЕМ КРИПТОГРАФИЧЕСКОЙ ИНФОРМАЦИИ (ТОКЕНОМ)

Інфармацыйныя тэхналогіі

ІНТЭРФЕЙС АБМЕНУ ІНФАРМАЦЫЯЙ З АПАРАТНА-ПРАГРАМНЫМ НОСЬБІТАМ КРЫПТАГРАФІЧНАЙ ІНФАРМАЦЫІ (ТОКЕНАМ)

(PKCS #11 v2.20:2004, IDT)

Издание официальное

БЗ 3-2009

Госстандарт

Минск

УДК 004.512(083.74)(476) МКС 35.040 КП 05 IDT

Ключевые слова: интерфейс обмена, аппаратно-программный носитель криптографической информации, токен

Предисловие

Цели, основные принципы, положения по государственному регулированию и управлению в области технического нормирования и стандартизации установлены Законом Республики Беларусь «О техническом нормировании и стандартизации».

1 ПОДГОТОВЛЕН научно-производственным республиканским унитарным предприятием «Белорусский государственный институт стандартизации и сертификации» (БелГИСС)

ВНЕСЕН Госстандартом Республики Беларусь

2 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ постановлением Госстандарта Республики Беларусь от 13 марта 2009 г. № 14

3 Настоящий стандарт идентичен стандарту PKCS #11 v2.20:2004 Cryptographic Token Interface Standard (Требования к представлению данных в аппаратно-программных носителях криптографической информации (токенах).

Стандарт разработан компанией RSA Laboratories в сотрудничестве с ведущими международными компаниями и специалистами в области систем безопасности.

Перевод с английского языка (en).

Экземпляр стандарта, на основе которого подготовлен настоящий государственный стандарт, имеется в Национальном фонде ТНПА.

Наименование настоящего стандарта изменено по отношению к наименованию PKCS #11 v2.20 с целью применения обобщающего понятия в соответствии с требованиями ТКП 1.5-2004 (04100).

Степень соответствия – идентичная (IDT)

4 ВЗАМЕН СТБ П 34.101.21-2006

Издан на русском языке

Содержание

ГОСУДАРСТВЕННЫЙ СТАНДАРТ РЕСПУБЛИКИ БЕЛАРУСЬ

Информационные технологии
ИНТЕРФЕЙС ОБМЕНА ИНФОРМАЦИЕЙ С АППАРАТНО-ПРОГРАММНЫМ НОСИТЕЛЕМ КРИПТОГРАФИЧЕСКОЙ ИНФОРМАЦИИ (ТОКЕНОМ)

Інфармацыйныя тэхналогіі
ІНТЭРФЕЙС АБМЕНУ ІНФАРМАЦЫЯЙ З АПАРАТНА-ПРАГРАМНЫМ НОСЬБІТАМ КРЫПТАГРАФІЧНАЙ ІНФАРМАЦЫІ (ТОКЕНАМ)

Information technology. Cryptographic Token Interface

Дата введения 2009-09-01

1 Введение

По мере того как криптография находит все большее применение и пользуется все большим признанием, становится все более очевидно, что, если она собирается стать настолько эффективной, насколько эффективна позволяющая ей это сделать технология, нужны совместимые стандарты. Даже если разработчики придут к определенному согласию относительно основных технологий криптографии, совместимость между различными программными решениями отнюдь не гарантирована. Для обеспечения совместимости требуется четкое соблюдение согласованных стандартов.

Для достижения этой цели исследовательский центр RSA Laboratories при сотрудничестве с представителями промышленности, науки и правительства разработал семейство стандартов под названием «Стандарты криптографии с открытым ключом», сокращенно – PKCS (Public-Key Cryptography Standards).

RSA Laboratories предлагает PKCS разработчикам компьютерных систем, в которых применяются методики с открытым ключом и связанные с ними технологии. Намерения RSA Laboratories состоят в совершенствовании и обновлении стандартов для разработчиков компьютерных систем с целью создания стандартов, которые будут приняты большинством, если не всеми разработчиками.

Роль RSA Laboratories в разработке стандартов состоит из четырех задач:

1) публикация документов, содержащих четкое описание стандартов;

2) выяснение мнений и учет предложений разработчиков и пользователей в отношении полезных и необходимых изменений и расширений;

3) публикация пересмотренных стандартов, если это необходимо;

4) создание руководств к программным решениям и/или эталонных программ.

В процессе разработки PKCS центр RSA Laboratories сохраняет за собой право принятия окончательной редакции любого документа, однако потенциальный вклад рецензентов также весьма важен. Основной целью RSA Laboratories является ускорение разработки официальных стандартов, а не конкурентная борьба в этой работе. Таким образом, когда документ PKCS принимается в качестве базового к официальному стандарту, RSA Laboratories отказывается от своего права «собственности» на этот документ, тем самым уступая дорогу процессу разработки открытых стандартов. RSA Laboratories может продолжать разрабатывать соответствующую документацию на условиях, описанных выше.

Документы и информацию по PKCS можно найти по адресу http://www.rsasecurity.com/rsalabs/PKCS/. На сайте rsasecurity.com «cryptoki» имеется возможность подписаться на электронную рассылку, которая связана непосредственно с обсуждением и разработкой PKCS #11. Чтобы подписаться на рассылку, отправьте сообщение по электронной почте на адрес majordomo@rsasecurity.com со строкой «subscribe cryptoki» в тексте письма.

Чтобы отказаться от подписки, отправьте сообщение на адрес majordomo@rsasecurity.com со строкой «unsubscribe cryptoki» в тексте письма.

Издание официальное

Комментарии относительно документов PKCS, запросы на регистрацию расширений стандартов, а также предложения по созданию дополнительных стандартов направляйте по адресу:

PKCS Editor
RSA Laboratories
174 Middlesex Turnpike
Bedford, MA 01730 USA
pkcs-editor@rsasecurity.com
http://www.rsasecurity.com/rsalabs/PKCS/

2 Вопросы, рассматриваемые стандартом

В настоящем стандарте задаются параметры программного интерфейса приложения (application programming interface, API), именуемого «Cryptoki», для устройств, содержащих криптографическую информацию и выполняющих криптографические функции. Cryptoki (произносится «криптоки» – сокращение от CRYPtographic TOKen Interface – интерфейс криптографического носителя) следует простому объектно-ориентированному подходу, решая задачи обеспечения технологической независимости (любой вид устройства) и совместного использования ресурсов (доступ различных приложений к различным устройствам), давая приложениям единое логическое представление устройства, именуемого «криптографическим носителем».

В настоящем стандарте определяются типы данных и функции, доступные приложению, которому требуются криптографические службы, использующие язык программирования ANSI C. Эти типы данных и функции будут, как правило, представляться через заголовочные файлы C поставщиком библиотеки Cryptoki. Групповые заголовочные файлы ANSI C доступны на интернет-странице PKCS. Там же можно будет найти настоящий документ и самый последний список опечаток для Cryptoki.

В дополнительных документах могут содержаться общие, не зависящие от языка сведения об интерфейсе Cryptoki и/или иные связующие звенья между Cryptoki и другими языками программирования.

Cryptoki отделяет приложение от подробных данных о криптографическом устройстве. Не нужно изменять приложение для установления связи с другими типами устройств или для его работы в ином окружении; таким образом, приложение является переносимым. Способы обеспечения такого изолированного использования приложения не объясняются в настоящем стандарте, хотя в нем, а также, возможно, и в других документах все-таки будут упоминаться некоторые общие вопросы поддержки многочисленных типов приложений.

Данная версия поддерживает определенные криптографические механизмы (алгоритмы). Кроме того, в дальнейшем могут быть добавлены новые механизмы без изменения общего интерфейса. Возможно, время от времени информация о дополнительных механизмах будет публиковаться в отдельных документах; поставщики носителей, возможно, также будут вводить собственные механизмы (хотя в целях обеспечения совместимости предпочтительной является регистрация с использованием процесса PKCS).

Интерфейс Cryptoki предназначен для криптографических устройств, связанных с одиночным пользователем, поэтому опущены некоторые свойства, которые могли бы быть включены в интерфейс общего назначения. Например, Cryptoki не содержит средств распознавания нескольких пользова-телей. Основное внимание уделяется ключам одного пользователя и, возможно, небольшому количеству связанных с ними сертификатов. Более того, сделан упор на криптографию. Хотя устройство и может выполнять полезные некриптографические функции, эти функции оставлены другим интерфейсам.

3 Нормативные ссылки

ANSI C ANSI/ISO Американский национальный стандарт на язык программирования – C. 1990

ANSI X9.31 Уполномоченный комитет по стандартизации X9. Цифровые сигнатуры, использующие криптографию с обратимым открытым ключем для идустрии финансовых услуг (rDSA), 1998

ANSI X9.42 Уполномоченный комитет по стандартизации X9. Криптография с открытым ключем для индустрии финансовых услуг: Соглашение по симметричным ключам с использованием дискретной логарифмической криптографии, 2003

ANSI X9.62 Уполномоченный комитет по стандартизации X9. Криптография с открытым ключем для идустрии финансовых услуг: Алгоритм цифровой подписи с использованием эллиптических кривых (ECDSA), 1998

ANSI X9.63 Уполномоченный комитет по стандартизации X9. Криптография с открытым ключем для идустрии финансовых услуг: Соглашение по ключам и по транспортировке ключей с использованием криптографии с эллиптическими кривыми, 2001

CC/PP W3C. Составные профили возможностей и предпочтений (CC/PP): Структура и словарь. Консорциум World-Wide Web, январь 2004. URL: http://www.w3.org/TR/CCPP-struct-vocab/

CDPD Ameritech Mobile Communications и др. Технические требования к системе сотовой цифровой пакетной передачи данных: Часть 406: Безопасность беспроводных соединений. 1993.

FIPS PUB 46–3 NIST. FIPS 46-3: Стандарт шифрования данных (DES). 25 октября 1999. URL: http://csrc.nist.gov/publications/fips/index.html

FIPS PUB 74 NIST. FIPS 74: Руководство по внедрению и эксплуатации стандарта шифрования данных NBS. 1 апреля 1981. URL: http://csrc.nist.gov/publications/fips/index.html

FIPS PUB 81 NIST. FIPS 81: Режимы использования DES. Декабрь 1980. URL: http://csrc.nist.gov/publications/fips/index.html

FIPS PUB 113 NIST. FIPS 113: Идентификация копьютерной информации. 30 мая 1985. URL: http://csrc.nist.gov/publications/fips/index.html

FIPS PUB 180-2 NIST. FIPS 180-2: Стандарт хэш-безопасности. 1 августа 2002. URL: http://csrc.nist.gov/publications/fips/index.html

FIPS PUB 186-2 NIST. FIPS 186-2: Стандарт цифровой подписи. 27 января 2000. URL: http://csrc.nist.gov/publications/fips/index.html

FIPS PUB 197 NIST. FIPS 197: Улучшенный стандарт шифрования (AES). 26 ноября 2001. URL: http://csrc.nist.gov/publications/fips/index.html

FORTEZZA CIPG NSA Программные продукты для обеспечения безопасности рабочих станций. Руководство по криптологическому интерфейсу программирования FORTEZZA, редакция 1.52, ноябрь 1995

GCS-API X/Open Company Ltd. Общие сервисы шифрования интерфейсов прикладного программирования (GCS-API). Основные требования. Проект 2, 14 февраля 1995

ISO/IEC 7816-1:1998 Идентификационные карточки. Карточки на интегральных схемах с контактами. Часть 1. Физические характеристики

ISO/IEC 8824-1:2002 Технологии информационные. Абстрактная система обозначения синтаксиса версии 1 (ASN.1). Часть 1. Технические условия на основную систему обозначений

ISO/IEC 8825-1:2002 Технологии информационные. Правила кодирования ASN.1. Часть 1. Технические условия на базовые правила кодирования (BER), канонические правила кодирования (CER) и различимые правила кодирования (DER)

ISO/IEC 9594-1:2001 Информационные технологии. Взаимодействие открытых систем. Директория. Обзор концепций, моделей и услуг

ISO/IEC 9594-8:2001 Информационные технологии. Взаимосвязь открытых систем. Справочник. Шифрование открытым ключом и качественные сертификатные структуры

ISO/IEC 9796-2:2002 Информационные технологии. Методы обеспечения безопасности. Схемы цифровой подписи, позволяющие восстанавливать сообщения. Часть 2. Механизмы, основанные на факторизации целого числа

Java MIDP Java Community Process. Профиль Java 2 Micro Edition для переносных информационных устройств. Ноябрь 2002. URL: http://jcp.org/jsr/detail/118.jsp

MeT-PTD MeT. Описание MeT PTD – Описание личного доверительного устройства, версия 1.0, февраль 2003. URL: http://www.mobiletransaction.org

PCMCIA Международная ассоциация производителей карт памяти для персональных компьютеров. Стандарт на PC-карты, издание 2.1, июль 1993

PKCS #1 RSA Laboratories. Информационные технологии. Стандарт криптографии RSA. v2.1, 14 июня 2002. URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html

PKCS #3 RSA Laboratories. Информационные технологии. Стандарт согласования ключей Диффи – Хеллмана. v1.4, ноябрь 1993. URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-3/index.html

PKCS #5 RSA Laboratories. Стандарт шифрования, основанный на паролях. v2.0, 25 марта 1999. URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-5/index.html

PKCS #7 RSA Laboratories. Информационные технологии. Стандарт синтаксиса криптографических сообщений. v1.5, ноябрь 1993. URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/index.html

PKCS #8 RSA Laboratories. Информационные технологии. Стандарт синтаксиса информации с индивидуальным ключом. v1.2, ноябрь 1993. URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-8/index.html

PKCS #11-C RSA Laboratories. Информационные технологии. Спецификация профиля соответствия, октябрь 2000. URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html

PKCS #11-P RSA Laboratories. Информационные технологии. Профили для мобильных устройств, июнь 2003. URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html

PKCS #12 RSA Laboratories. Информационные технологии. Стандарт синтаксиса обмена персональной информацией. v1.0, июнь 1999. URL: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-12/index.html

RFC 1319: Алгоритм профиля сообщений MD2. RSA Laboratories, B. Kaliski., апрель 1992. URL: http://ietf.org/rfc/rfc1319.txt

RFC 1321: Алгоритм профиля сообщений MD5. MIT Laboratory for Computer Science and RSA Data Security, Inc., R. Rivest., апрель 1992. URL: http://ietf.org/rfc/rfc1321.txt

RFC 1421: Совершенствование конфиденциальности электронной почты в сети Интернет: Часть 1: Шифрование сообщений и процедуры аутентификации. IAB IRTF PSRG, IETF PEM WG, J. Linn., февраль 1993. URL: http://ietf.org/rfc/rfc1421.txt

RFC 2045: Многоцелевые расширения электронной почты в Интернете (MIME) Часть 1: Формат содержимого интернет-сообщений. Freed N. и N. Borenstein, ноябрь 1996. URL: http://ietf.org/rfc/rfc2045.txt

RFC 2246: TLS протокол версии 1.0. Certicom, T. Dierks & C. Allen., январь 1999. URL: http://ietf.org/rfc/rfc2246.txt

RFC 2279: UTF-8, преобразование формата из ISO 10646 Alis Technologies, F. Yergeau., январь 1998. URL: http://ietf.org/rfc/rfc2279.txt

RFC 2534: Медиа-возможности дисплея, принтера и факса. Masinter, L., Wing, D., Mutz, A. и K. Holtman, март 1999. URL: http://ietf.org/rfc/rfc2534.txt

RFC 2630: Синтаксис шифрования сообщений. R. Housley., июнь 1999. URL: http://ietf.org/rfc/rfc2630.txt

RFC 2743: Общий интерфейс служб безопасности прикладных программ версии 2, обновление 1. RSA Laboratories, J. Linn., январь 2000. URL: http://ietf.org/rfc/rfc2743.txt

RFC 2744: API для служб безопасности версии 2: C-связи. Iris Associates, J. Wray., январь 2000. URL: http://ietf.org/rfc/rfc2744.txt

SEC 1 Стандарты группы по эффективной криптографии (SECG). Стандарты по эффективной криптографии (SEC) 1: Шифрование эллиптическими кривыми. Версия 1.0, 20 сентября 2000

SEC 2 Стандарты группы по эффективной криптографии (SECG). Стандарты по эффективной криптографии (SEC) 2: Рекомендуемые доменные параметры эллиптических кривых. Версия 1.0, 20 сентября 2000

TLS IETF. RFC 2246 – TLS-протокол версии 1.0, январь 1999. URL: http://ietf.org/rfc/rfc2246.txt

WIM WAP. Беспроводной модуль индентификации. – WAP-260-WIM-20010712-a. Июль 2001. URL: http://www.wapforum.org/

WPKI WAP. Беспроводной PKI. – WAP-217-WPKI-20010424-a. Апрель 2001. URL: http://www.wapforum.org/

WTLS WAP. Беспроводной транспортный протокол версии – WAP-261-WTLS-20010406-a. Апрель 2001. URL: http://www.wapforum.org/

X.500 ITU-T. Информационные технологии. Взаимодействие открытых систем. Справочник: Общее описание принципов, моделей и услуг. Февраль 2001. Идентичен ISO/IEC 9594-1

X.509 ITU-T. Информационные технологии. Взаимодействие открытых систем. Справочник: Открытый ключ и атрибуты структуры сертификатов. Март 2000. Иднтичен ISO/IEC 9594-8

X.680 ITU-T. Информационные технологии – Абстрактная синтаксическая нотация версии один (ASN.1): Спецификация основной нотации. Июль 2002. Идентичен ISO/IEC 8824-1

X.690 ITU-T. Информационные технологии. Правила кодирования ASN.1: Технические требования к базовым правилам кодирования (BER), каноническим правилам кодирования (CER) и отличительным правилам кодирования (DER). Июль 2002. Идентичен ISO/IEC 8825-1

4 Термины и определения

В настоящем стандарте применяются следующие термины с соответствующими определениями:

API (Application Programming Interface): Программный интерфейс приложения.

Приложение (Application): Любая компьютерная программа, обращающаяся с запросом к интерфейсу Cryptoki.

ASN.1 (Abstract Syntax Notation One): Система обозначений общего синтаксиса один, в соответствии с X.680.

Атрибут (Attribute): Характеристика объекта.

BATON: Блочный шифр BATON, предложенный MISSI.

BER (Basic Encoding Rules): Базовые правила кодирования, в соответствии с X.690.

CAST: Симметричный блочный шифр компании Entrust Technologies.

CAST3: Симметричный блочный шифр компании Entrust Technologies.

CAST5: Иное наименование симметричного блочного шифра компании Entrust Technologies CAST128. CAST128 − предпочтительное наименование.

CAST128: Симметричный блочный шифр компании Entrust Technologies.

CBC (Cipher-Block Chaining): Режим цепочки шифроблоков, в соответствии с документом FIPS PUB 81.

CDMF (Commercial Data Masking Facility): Средство закрытия коммерческих данных, метод блочного шифрования, разработанный корпорацией IBM на основе стандарта DES.

Сертификат (Certificate): Сообщение с подписью, связывающее имя субъекта с открытым ключом или имя субъекта с набором атрибутов.

CMS (Cryptographic Message Syntax): Синтаксис криптографического сообщения (см. RFC 2630).

Криптографическое устройство (Cryptographic Device): Устройство, в котором хранится криптографическая информация и которое, возможно, выполняет криптографические функции. Оно может быть реализовано в виде смарт-карты, смарт-диска, PCMCIA-карты или через какую-либо иную технологию, включая исключительно программную реализацию.

Cryptoki (CRYPtographic TOKen Interface): Интерфейс криптографического носителя, описание которого дается в настоящем стандарте.

Библиотека Cryptoki (Cryptoki library): Библиотека, реализующая функции, приведенные в настоящем стандарте.

DER (Distinguished Encoding Rules): Характерные правила кодирования, в соответствии со стандартом X.690.

DES (Data Encryption Standart): Стандарт шифрования данных, в соответствии с документом FIPS PUB 46-3.

DSA (Digital Signature Algorithm): Алгоритм электронной цифровой подписи, в соответствии с документом FIPS PUB 186-2.

EC (Elliptic Curve): Эллиптическая кривая.

ECB (Electronic Codebook): Электронная кодовая книга, в соответствии с документом FIPS PUB 81

ECDH (Elliptic Curve Diffie-Hellman): Эллиптическая кривая Диффи – Хеллмана.

ECDSA (Elliptic Curve DSA): Алгоритм электронной цифровой подписи на основе эллиптической кривой, в соответствии со стандартом ANSI X9.62.

ECMQV (Elliptic Curve Menezes-Qu-Vanstone): Эллиптическая кривая Менезеса – Ку – Вэнстоуна.

FASTHASH: Алгоритм хэширования FASTHASH, предложенный MISSI.

IDEA (International Data Encryption Algorithm): Международный алгоритм шифрования данных. Симметричный блочный шифр компании Ascom Systec.

IV (Initialization Vector): Вектор инициализации.

JUNIPER: Блочный шифр JUNIPER, предложенный MISSI.

KEA (Key Exchange Algorithm): Алгоритм обмена ключами, предложенный MISSI.

LYNKS: Смарт-карта производства компании SPYRUS.

MAC (Message Authentication Code): Код аутентификации сообщения.

MD2 (Message-Digest Algorithm): Алгоритм получения списка сообщения, алгоритм хэширования MD2 компании RSA Security, в соответствии с документом RFC 1319.

MD5 (Message-Digest Algorithm): Алгоритм получения списка сообщения, алгоритм хэширования MD5 компании RSA Security, в соответствии с документом RFC 1321.

Механизм (Mechanism): Процесс реализации криптографических операций.

MQV: Менезес – Ку – Вэнстоун.

OAEP (Optimal Asymmetric Encryption Padding): Оптимальное заполнение при асимметричном шифровании RSA.

Объект (Object): Элемент, содержащийся в носителе. Это могут быть данные, сертификат или ключ.

PIN (Personal Identification Number): Личный идентификационный номер.

PKCS (Public-Key Cryptography Standards): Стандарты шифрования с открытым ключом.

PRF (Pseudo random function): Псевдослучайная функция.

PTD (Personal Trusted Device): Персональное доверенное устройство, в соответствии с документом МеТ-PTD.

RSA: Криптосистема с открытым ключом компании RSA Security.

RC2: Симметричный блочный шифр RC2 компании RSA Security.

RC4: Симметричный поточный шифр RC4 компании RSA Security.

RC5: Симметричный поточный шифр RC5 компании RSA Security.

Reader: Считывающее устройство, средство обмена информацией.

Сеанс (Session): Логическое соединение между приложением и носителем.

SET (Secure Electronic Transaction): Протокол защищенных электронных транзакций.

SHA-1 (Secure Hash Algorithm – 1): (Исправленный) алгоритм безопасного хэширования со 160-битным шифрованием, в соответствии с документом FIPS PUB 180-2.

SHA-256 (Secure Hash Algorithm – 256): Алгоритм безопасного хэширования с 256-битным шифрованием, в соответствии с документом FIPS PUB 180-2.

SHA-384 (Secure Hash Algorithm – 384): Алгоритм безопасного хэширования с 384-битным шифрованием, в соответствии с документом FIPS PUB 180-2.

SHA-512 (Secure Hash Algorithm – 512): Алгоритм безопасного хэширования с 512-битным шифрованием, в соответствии с документом FIPS PUB 180-2.

Слот (Slot): Логическое считывающее устройство, которое может иметься у носителя.

SKIPJACK: Блочный шифр SKIPJACK, предложенный MISSI.

SSL (Secure Sockets Layer 3.0 protocol): Протокол безопасных соединений SSL 3.0.

Имя субъекта (Subject Name): В соответствии с протоколом X.500 – уникальное имя единицы, к которой прикрепляется ключ.

SO (Security Officer): Пользователь, являющийся ответственным за безопасность.

TLS (Transport Layer Security): Протокол безопасности транспортного уровня.

Token: Логическое представление криптографического устройства, параметры которого заданы настоящим документом о Cryptoki.

Пользователь (user): Лицо, которое работает с приложением, использующим интерфейс Cryptoki.

UTF-8 (Universal Character Set (UCS) Transformation Format (UTF): Формат преобразования (UTF) универсального набора символов (UCS), который представляет строки форматов ISO 10646 и UNICODE различным количеством октетов.

WIM (Wireless Identification Module): Идентификационный модуль для беспроводной связи.

WTLS (Wireless Transport Layer Security): Протокол безопасности транспортного уровня для беспроводной связи.

5 Условные знаки и сокращения

В настоящем стандарте используются условные знаки, приведенные в таблице 1.

Таблица 1 – Условные знаки

Условный знак	Определение
N/A	Не применяется
R/O	Только чтение
R/W	Чтение/запись

В настоящем стандарте используются префиксы, приведенные в таблице 2.

Таблица 2 – Префиксы

Префикс	Описание
C_	Функция
CK_	Тип данных или общая константа
CKA_	Атрибут
CKC_	Тип сертификата
CKD_	Функция деривации ключа
CKF_	Бит-флаг
CKG_	Функция генерации маски
CKH_	Тип аппаратного средства
CKK_	Тип ключа
CKM_	Тип механизма
CKN_	Уведомление

Окончание таблицы 2

Префикс	Описание
CKO_	Класс объекта
CKP_	Псевдослучайная функция
CKS_	Состояние сеанса
CKR_	Возвращаемое функцией значение
CKU_	Тип пользователя
CKZ_	Источник параметра «salt»/параметра шифрования
H	Идентификатор
Ul	CK_ULONG
P	Указатель
Pb	Указатель на CK_BYTE
Ph	Указатель на идентификатор
Pul	Указатель на CK_ULONG

Интерфейс Cryptoki основан на типах языка ANSI C и определяет следующие типы данных:

```
/* an unsigned 8-bit value */
typedef unsigned char CK_BYTE;

/* an unsigned 8-bit character */
typedef CK_BYTE CK_CHAR;

/* an 8-bit UTF-8 character */
typedef CK_BYTE CK_UTF8CHAR;

/* a BYTE-sized Boolean flag */
typedef CK_BYTE CK_BBOOL;

/* an unsigned value, at least 32 bits long */
typedef unsigned long int CK_ULONG;

/* a signed value, the same size as a CK_ULONG */
typedef long int CK_LONG;

/* at least 32 bits; each bit is a Boolean flag */
typedef CK_ULONG CK_FLAGS;
```

Cryptoki использует также указатели на некоторые из этих типов данных, а также на тип `void`, которые зависят от реализации. Вот эти типы указателей:

```
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 также определяет указатель на CK_VOID_PTR, который зависит от реализации:

```
CK_VOID_PTR_PTR /* Pointer to a CK_VOID_PTR */
```

Кроме того, Cryptoki определяет пустой указатель типа C-style, который отличается от любого действительного указателя:

```
NULL_PTR /* A NULL pointer */
```

Следовательно, многие типы данных и указателей будут каким-то образом отличаться в разных окружениях (например, CK_ULONG иногда будет равно 32 битам, а иногда, возможно, и 64 битам). Однако эти подробности не повлияют на приложение, при условии, что оно скомпилировано с использованием заголовочных файлов Cryptoki, единообразных в пределах библиотеки Cryptoki, с которой связано приложение.

Все числа и значения, представленные в настоящем стандарте, являются десятичными, за исключением случаев, когда перед числом стоит сочетание «0x», являющееся указателем на последующую шестнадцатеричную величину.

Тип данных **CK_CHAR** содержит перечисленные в таблице 3 символы, взятые из языка ANSI C.

Таблица 3 – Набор символов

Категория	Символы
Буквы	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
Цифры	0 1 2 3 4 5 6 7 8 9
Графические символы	! “ # % & ‘ ( ) * + , - . / : ; < = > ? [\ ] ^ _ { | } ~
Пустой символ	‘ ‘

Тип данных **CK_UTF8CHAR** содержит символы Unicode, закодированные посредством UTF-8 в соответствии с документом RFC2279. Формат UTF-8 позволяет интернационализацию символов при поддержке обратной совместимости с определением Local String (локальная строка) из версии 2.01 PKCS #11.

В Cryptoki тип данных **CK_BBOOL** – это булев тип, который может принимать значения «истинно» или «ложно». Нулевое значение указывает на «ложно», а значение, не равное нулю, означает «истинно». Точно так же отдельный бит-флаг **CKF**_... может быть выставлен (истинно) или не выставлен (ложно). Для удобства Cryptoki определяет следующие макросы для использования со значениями типа **CK_BBOOL**:

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

В целях обеспечения обратной совместимости заголовочные файлы для данной версии Cryptoki также содержат следующие определения TRUE и FALSE (CK_DISABLE_TRUE_FALSE может быть выставлен поставщиком приложения):

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

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

6 Общие сведения

6.1 Введение

Портативные вычислительные устройства, такие как смарт-карта, карта PCMCIA и смарт-дискета, являются идеальными инструментами для реализации криптографической технологии с открытым ключом, так как предоставляют возможность безопасного хранения компонента «секретный ключ» пары открытый ключ/секретный ключ под контролем одного пользователя. При наличии такого устройства криптографическое приложение не выполняет криптографические операции самостоятельно, а использует данное устройство для выполнения этих операций, при этом уязвимая информация, такая как секретный ключ, никогда не раскрывается. По мере того как разрабатывается все большее количество приложений для использования технологии криптографии с открытым ключом, все более важным становится вопрос разработки стандартного программного интерфейса для таких устройств. Настоящий стандарт направлен на разрешение данного вопроса.

6.2 Цели проекта

С самого начала Cryptoki разрабатывался как интерфейс между приложениями и всеми видами портативных криптографических устройств, таких как устройства, основанные на использовании смарт-карт, карт PCMCIA и смарт-дискет. Уже существуют стандарты (де-факто или официально), регламентирующие сопряжение этих устройств на определенном уровне. Например, четко определены механические характеристики и электрические соединения, так же как методы подачи команд и получения результатов (см. например, ISO 7816 или спецификации PCMCIA).

Оставалось определить конкретные команды для выполнения шифрования. Недостаточно было бы просто определить наборы команд для каждого вида устройств, так как это не решало бы общую проблему создания интерфейса *приложения*, независимого от конкретного устройства. Решение данного вопроса все еще представляется как дальняя перспектива, которая определенно внесла бы вклад в обеспечение совместимости. Основной целью Cryptoki было создание программного интерфейса низшего уровня, который абстрагируется от подробных характеристик устройств и представляет приложению общую модель криптографического устройства, именуемого «криптографический носитель» (или просто «носитель»).

В качестве вторичной цели рассматривалось совместное использование ресурсов. По мере того как все большей популярностью пользуются многоцелевые операционные системы для настольных компьютеров, одно и то же устройство должно совместно использоваться более чем одним приложением. Кроме того, приложение должно иметь возможность взаимодействовать более чем с одним устройством в определенный момент времени.

Задачей Cryptoki не является быть общим интерфейсом для криптографических операций и служб обеспечения безопасности, хотя конечно же можно наладить такие операции и службы при помощи функций, предлагаемых интерфейсом Cryptoki. Cryptoki предназначен для дополнения таких появляющихся и развивающихся интерфейсов, как «Generic Security Services Application Programming Interface» («Общий программный интерфейс для приложения с функциями служб безопасности», RFC 2743 и RFC 2744) (RFC 2743 и RFC 2744), а также «Generic Cryptographic Service API» («Общий программный интерфейс приложения с фукнциями криптографической службы», GCS-API) из X/Open, а не для соперничества с ними.

6.3 Общая модель

Обобщенная модель интерфейса Cryptoki показана на рисунке 1. Модель начинается с одного или нескольких приложений, которые необходимы для выполнения определенных криптографических операций, и заканчивается одним или несколькими криптографическими устройствами, на которых фактически и выполняются некоторые или все операции. Пользователь может быть, а может и не быть связан с приложением.

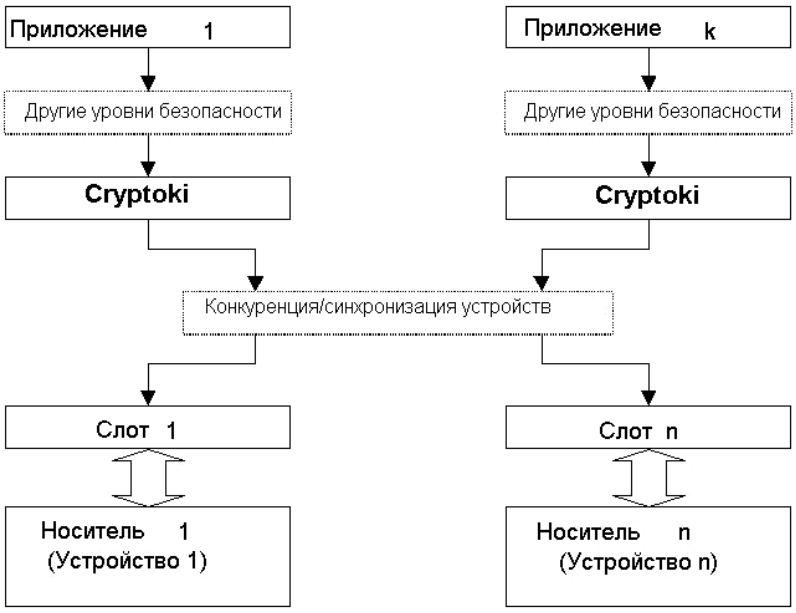

Рисунок 1 – Общая модель Cryptoki

Cryptoki предоставляет интерфейс для одного или нескольких криптографических устройств, активизированных в системе через несколько слотов. Каждый слот, соответствующий интерфейсу физического считывающего или иного устройства, может содержать носитель. Носитель обычно представлен в слоте, когда в считывающем устройстве представлено криптографическое устройство. Конечно, поскольку Cryptoki дает логическое представление слотов и носителей, возможны и иные физические интерпретации. Возможно, чтобы несколько слотов использовали одно физическое считывающее устройство. Дело в том, что система имеет некоторое количество слотов, а приложения могут соединяться с носителями через любой или все эти слоты.

Криптографическое устройство может выполнять некоторые криптографические операции, следуя определенному набору команд; такие команды обычно передаются через стандартные драйверы устройств (например, через службы карты PCMCIA или через службы разъема). Cryptoki создает каждому криптографическому устройству логическое представление, соответствующее логическому представлению любого другого устройства, независимо от технологии реализации. Таким образом, приложению не нужно напрямую связываться с драйверами устройств (или даже знать, какие устройства задействованы); Cryptoki скрывает данную информацию. Лежащее в основе устройство может быть полностью реализовано при помощи программных средств (например, процесс, выполняемый на сервере) – не нужны какие-либо специальные аппаратные устройства.

Вероятна реализация Cryptoki в качестве библиотеки, поддерживающей функции в интерфейсе, с привязкой приложений к этой библиотеке. Приложение может быть привязано к Cryptoki напрямую; как альтернативрный вариант, Cryptoki может стать так называемой совместно используемой библиотекой (или библиотекой динамической компоновки), в таком случае приложение будет привязано к библиотеке динамически. Совместно используемые библиотеки достаточно просты, чтобы создавать их в таких операционных системах, как Microsoft Windows или OS/2, они также и могут без всяких сложностей быть сформированы в системах UNIX и DOS.

Динамический подход, конечно, имеет преимущества, так как становятся доступными новые библиотеки, однако, исходя из перспектив безопасности, существуют некоторые недостатки. В частности, если библиотеку легко заменить, существует вероятность того, что атакующий сможет подменить существующую библиотеку и перехватить PIN пользователя. Следовательно, с точки зрения безопасности прямая привязка в целом предпочтительнее, хотя технология электронной подписи может противостоять многим рискам динамической увязки. В любом случае, независимо от того, является ли привязка прямой или динамической, программный интерфейс между приложением и библиотекой Cryptoki остается тем же.

Виды устройств и возможности, которые поддерживаются ими, будут зависеть от конкретной библиотеки Cryptoki. Настоящий стандарт определяет только интерфейс библиотеки, но не ее свойства. В частности, не все библиотеки будут поддерживать все механизмы (алгоритмы), определенные для данной библиотеки (так как ожидается, что не все носители будут поддерживать все механизмы), а библиотеки, скорее всего, будут поддерживать только одну подгруппу из всех видов доступных криптографических устройств. Конечно, чем больше видов устройств, тем лучше, и ожидается, что библиотеки будут разрабатываться с поддержкой большого количества носителей, а не только определенной группы, связанной общим поставщиком. Ожидается, что по мере разработки приложений, связанных с Cryptoki, появятся стандартные библиотеки и профили носителей.

6.4 Логическое представление носителя

Логическое представление носителя в Cryptoki состоит в том, что носитель – это устройство, хранящее объекты и способное выполнять криптографические функции. Cryptoki различает три класса объектов: данные, сертификаты и ключи. Объект данных задается приложением. В объекте сертификата хранится сертификат. В объекте ключа хранится криптографический ключ. Этот ключ может быть открытым, частным/секретным или секретным, при этом у каждого из этих типов ключа есть подтипы для использования в определенных механизмах. Описанное логическое представление показано на рисунке 2.

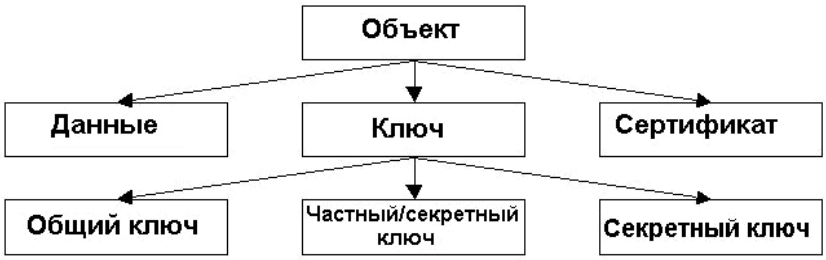

Рисунок 2 – Иерархия объектов

Объекты также классифицируются по сроку их существования и явности. Объекты носителя являются явными для всех приложений, связанных с носителем, у которых есть достаточные права, и остаются на носителе даже после завершения сеансов (соединений между приложением и носителем) и после удаления носителя из его слота. Сеансовые объекты носят более временный характер: когда бы и как бы ни прекращался сеанс, все сеансовые объекты, созданные этим сеансом, автоматически ликвидируются. Кроме того, сеансовые объекты явны только для создавшего их приложения.

В более подробной классификации определяются требования к доступу. Приложениям не нужно получать доступ к носителю, чтобы просмотреть открытые объекты, однако, чтобы просмотреть частные/секретные объекты, пользователь должен пройти процедуру аутентификации для носителя посредством PIN или каких-либо иных методов, зависящих от носителя (например, биометрическое устройство).

Более подробная классификация по доступу к объектам приводится в таблице 6.

Носитель может создавать и уничтожать объекты, управлять ими и производить их поиск. Также он может выполнять криптографические операции над объектами. В носитель может быть встроен генератор случайных чисел.

Важно делать различие между логическим представлением носителя и его фактической реализацией, так как не все криптографические устройства будут иметь аналогичное понятие «объекта» и не все они будут выполнять все виды криптографических функций. Многие устройства просто будут иметь фиксированные места для ключей фиксированного алгоритма с ограниченным объемом памяти, и эти устройства смогут выполнять ограниченный набор операций. Задача Cryptoki состоит в транслировании данных особенностей в логическое представление с отображением атрибутов на фиксированные элементы хранения и т. д. Не всем библиотекам и носителям Cryptoki нужно поддерживать все виды объектов. Ожидается, что будут разработаны стандартные профили с указанием набора алгоритмов, которые должны поддерживаться.

Атрибуты – это характеристики, по которым различаются частные случаи объекта. В Cryptoki есть общие атрибуты, указывающие, является объект секретным или открытым. Также есть атрибуты, которые используются исключительно с определенным типом объекта, такие как коэффициент (модуль) или показатель степени (экспонент) для ключей RSA.

6.5 Пользователи

Данная версия Cryptoki различает два типа пользователей носителя. Один тип – это ответственный за безопасность (Security Officer, SO). Второй тип – обычный пользователь. Только обычному пользователю предоставляется доступ к секретным объектам на носителе, при этом такой доступ дается только после того, как обычный пользователь прошел процедуру аутентификации. Некоторые носители могут требовать от пользователя аутентификации до того, как появится возможность выполнить какую-либо криптографическую функцию на носителе, и независимо от того, связана ли эта функция с использованием секретных объектов. Роль SO заключается в том, чтобы инициализировать носитель и активировать PIN обычного пользователя (или задать иным путем, находящимся вне пределов возможностей настоящей версии Cryptoki, способ аутентификации обычного пользователя), а также, возможно, чтобы управлять некоторыми открытыми объектами. Обычный пользователь не может зарегистрироваться до того, как SO активирует PIN этого обычного пользователя.

За исключением поддержки двух типов пользователей, Cryptoki не охватывает взаимоотношений между SO и сообществом пользователей. В частности, SO и обычный пользователь могут быть одним и тем же лицом, а могут быть и разными лицами, но эти вопросы не входят в сферу действия настоящего стандарта.

В отношении PIN, вводимых через приложение, Cryptoki ограничивается предположением, что это строки переменной длины, состоящие из символов, перечисленных в таблице 3. Любое преобразование в соответствии с требованиями устройства возлагается на библиотеку. За пределы компетенции Cryptoki выходят следующие вопросы:

– любое заполнение свободного поля PIN;

– способ генерации PIN (пользователем, приложением или как-либо иначе).

Значения PIN, генерируемые средствами вне приложения (например, при вводе PIN через интерфейс «PINpad» на носителе), являются еще более абстрактными. Cryptoki располагает более чем достаточной информацией о том, как ожидать (если необходимо) доставку и использование такого PIN.

6.6 Приложения и использование ими интерфейса Cryptoki

Для Cryptoki приложение состоит из одного адресного пространства и всех работающих в нем потоков управления. Приложение становится приложением Cryptoki, запрашивая функцию Cryptoki **C_Initialize** (см. 11.4), которая имеется в одном из его потоков; после такого запроса приложение может вызвать другие функции Cryptoki. Когда приложение выполняется с использованием Cryptoki, оно вызывает функцию Cryptoki **C_Finalize** (см. 11.4) и прекращает быть приложением Cryptoki.

6.6.1 Приложения и процессы

В целом, для большинства платформ все, изложенное в предыдущем параграфе, означает, что приложение состоит из одного процесса.

Рассмотрим процесс **P** в операционной системе UNIX, который становится приложением Cryptoki, вызывая **C_Initialize**, а затем использует системный вызов `fork()` для создания порожденного процесса **C**. Так как **P** и **C** имеют разные адресные пространства (или будут иметь разные адресные пространства, когда один из них выполнит операцию «запись», если операционная система следует принципу копирования при записи), они не являются частью одного и того же приложения. Следовательно, если процессу **C** нужно использовать интерфейс Cryptoki, ему нужно выполнить собственный вызов **C_Initialize**. Более того, если процессу **C** нужно зарегистрироваться на носителе (ях), доступ к которым он будет получать через Cryptoki, ему нужно зарегистрироваться на них, *даже если **P** уже зарегистрировался,* так как **P** и **C** – это абсолютно разные приложения.

В данном конкретном случае (**C** является порождением процесса, являющегося приложением Cryptoki) режим работы Cryptoki не задается, когда **C** пытается его использовать без собственного вызова **C_Initialize**. В идеальном случае, при такой попытке устройство выдаст значение CKR_CRYPTOKI_NOT_INITIALIZED, однако в зависимости от того, как работает `fork()`, настойчивые попытки получить это возвращаемое значение могут оказывать негативное влияние на производительность библиотек. Следовательно, режим работы Cryptoki в данной ситуации остается незаданным. Определенно приложение *не* должно пытаться воспользоваться какими-либо потенциальными «ярлыками», которые могли бы быть (а могли бы и не быть) доступны из-за этого.

В сценарии, описанном выше, процесс **C** фактически должен вызвать **C_Initialize** независимо от того, нуждается ли он в использовании Cryptoki; в случае, когда ему не нужно использовать Cryptoki, он должен вызвать **C_Finalize** сразу после этого. Данный вариант (когда порожденный процесс сразу же вызывает **C_Initialize**, а затем **C_Finalize**, если родительский процесс использует Cryptoki) считается нормальной программной практикой Cryptoki, так как он может предотвратить избыточное дублирование ресурсов, которые были созданы во время вызова `fork()`; однако Cryptoki не требует обязательного выполнения данных действий.

6.6.2 Приложения и потоки

Некоторые приложения будут получать доступ к библиотеке Cryptoki в многопоточном режиме. Cryptoki позволяет приложениям предоставлять информацию библиотекам, чтобы они могли обеспечить соответствующую поддержку многопоточной работы. В частности, когда приложение инициализирует библиотеку Cryptoki при помощи запроса к **C_Initialize**, оно может указать один из четырех возможных вариантов многопоточной работы библиотеки:

1. Приложение может указать, что оно не будет обращаться к библиотеке одновременно из нескольких потоков, и поэтому библиотеке не нужно выполнять какого-либо рода блокировку в целях безопасности потоков.

2. Приложение может указать, что оно *будет* обращаться к библиотеке одновременно из нескольких потоков, и поэтому библиотека должна обеспечить использование базовых элементов синхронизации собственной операционной системы для обеспечения работы с надлежащей безопасностью потоков.

3. Приложение может указать, что оно *будет* обращаться к библиотеке одновременно из нескольких потоков, а библиотека должна использовать предоставляемые приложением базовые элементы синхронизации для обеспечения работы с надлежащей безопасностью потоков.

4. Приложение может указать, что оно *будет* обращаться к библиотеке одновременно через несколько потоков, а библиотека должна использовать либо базовые элементы синхронизации собственной операционной системы, либо предоставляемые приложением базовые элементы синхронизации для обеспечения работы с надлежащей безопасностью потоков.

3-й и 4-й типы режимов работы, перечисленные выше, – это режимы для многопоточных приложений, которые не используют потоковую модель собственной операционной системы. Элементы синхронизации, предоставляемые приложением, состоят из четырех функций управления флагами (mutex – mutual exclusion, взаимное исключение – *механизм предотвращения доступа к общему ресурсу*) в потоковой модели приложения. Флажные объекты – это простые объекты, которые могут быть в одном из двух состояний в отдельный момент времени: заблокированном или разблокированном. Если поток отдает команду на блокировку флага, который уже заблокирован, то сам этот поток блокируется (ожидает) до тех пор, пока флаг не будет разблокирован, после чего поток блокирует его снова и вызов возвращается. Если более чем один поток пытается блокировать один и тот же флаг, а этот флаг становится разблокированным, то именно один из этих потоков сможет заблокировать флаг и получить возврат вызова (остальные блокирующие потоки будут продолжать блокировку, ожидая своей очереди).

Для получения дополнительной информации по видам флагов в Cryptoki см. 9.7.

Кроме предоставления библиотеке Cryptoki в момент инициализации описанной выше информации по управлению потоками, приложение также может указывать, могут или не могут потоки приложения, выполняющие вызовы библиотеки, использовать вызовы собственной операционной системы для порождения новых потоков.

6.7 Сеансы

Для Cryptoki требуется, чтобы приложение открыло не менее одного сеанса связи с носителем, чтобы получить доступ к объектам и функциям носителя. Сеанс обеспечивает логическое соединение между приложением и носителем. Сеанс может быть сеансом чтения/записи (read/write, R/W) или сеансом только чтения (read-only, R/O). Чтение/запись и только чтение – это термины, характеризующие тип доступа к объектам носителя, а не к объектам сеанса. При обоих типах сеансов приложение может создать, прочитать, записать и уничтожить объекты сеанса, а также прочитать объекты носителя. Однако только в сеансе чтения/записи приложение может создавать, изменять и уничтожать объекты носителя.

После того как приложение открыло сеанс, оно получает доступ к открытым объектам носителя. Все потоки отдельного приложения имеют доступ к одним и тем же сеансам и объектам тех же самых сеансов. Чтобы получить доступ к секретным объектам носителя, обычный пользователь должен зарегистрироваться в нем и пройти аутентификацию.

Когда сеанс закрывается, любые объекты сеанса, которые создавались в рамках данного сеанса, уничтожаются. Это относится даже к тем объектам сеанса, которые «используются» другими сеансами. Это означает, что, если у одного приложения открыто несколько сеансов связи с носителем и оно использует один из них для создания объекта сеанса, то этот объект сеанса будет явным для всех сеансов приложения. Однако как только сеанс, который использовался для создания объекта, будет закрыт, данный объект будет уничтожен.

Cryptoki поддерживает несколько сеансов на нескольких носителях. Приложение может иметь один или несколько сеансов с одним или несколькими носителями. Вообще, у носителя может быть несколько сеансов с одним или более приложениями. Отдельный носитель может давать приложению возможность вести ограниченное количество сеансов или только ограниченное количество сеансов чтения/записи.

Открытый сеанс может быть в одном из нескольких состояний. Состояние сеанса определяет разрешенный ему уровень доступа к объектам и функциям. Описание состояний сеансов приводится в 6.7.1 и 6.7.2.

6.7.1 Состояния сеанса только для чтения

Сеанс только для чтения может быть в одном из двух состояний, как это показано на рисунке 3. При первоначальном открытии сеанса он будет или в состоянии «Открытый сеанс только для чтения» (если до этого приложение не открывало сеансов, в которых зарегистрировался пользователь), или в состоянии «Сеанс пользовательских функций только для чтения» (если приложение уже поддерживает ранее открытый сеанс, в котором зарегистрировался пользователь). Внимание! Сеансов SO только для чтения не существует.

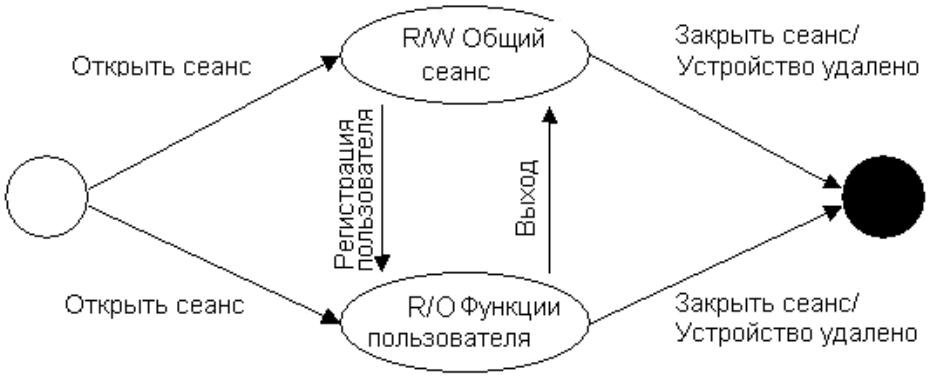

Рисунок 3 – Состояния сеанса только для чтения

В таблице 4 приводится описание состояний сеанса.

Таблица 4 – Состояния сеанса только для чтения

Состояние	Описание
Общий сеанс только для чтения	Приложение открыло сеанс только для чтения. У приложения есть доступ только для чтения к открытым объектам носителя и доступ чтения/записи к открытым объектам сеанса
Сеанс пользовательских функций только для чтения	Обычный пользователь прошел аутентификацию на носителе. У приложения есть доступ только для чтения ко всем объектам носителя (открытым и секретным) и доступ чтения/записи ко всем объектам сеанса (открытым и секретным)

6.7.2 Состояния сеанса чтения/записи

Сеанс чтения/записи может быть в одном из двух состояний, как это показано на приведенном ниже рисунке. При открытии сеанса он будет либо в состоянии «Общий сеанс чтения/записи» (если до этого приложение не открывало сеансов, в которых имела место регистрация), либо в состоянии «Сеанс пользовательских функций чтения/записи», либо в состоянии «Сеанс функций SO чтения/записи» (если приложение уже имеет открытый сеанс, в котором зарегистрировался SO).

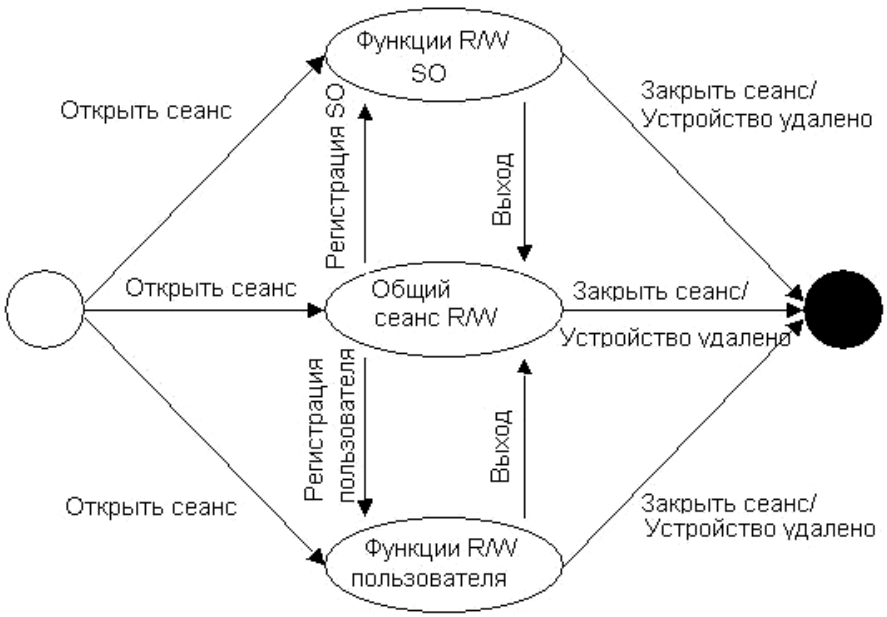

Рисунок 4 – Состояния сеанса чтение/запись

В таблице 5 приводится описание состояний сеанса.

Таблица 5 – Состояния сеанса чтение/запись

Состояние	Описание
Открытый сеанс чтения/ записи	Приложение открыло сеанс чтения/записи. У приложения есть доступ чтения/записи ко всем открытым объектам
Сеанс функций SO чтения/ записи	Офицер безопасности прошел аутентификацию на носителе. У приложения есть доступ чтения/записи только к открытым объектам носителя, но не к секретным объектам. SO может ввести PIN обычного пользователя
Сеанс пользовательских функций чтения/записи	Обычный пользователь прошел аутентификацию на носителе. У приложения есть доступ чтения/записи ко всем объектам

6.7.3 Разрешенные виды доступа сеансов к объектам

В таблице 6 приведена краткая информация о видах доступа каждого типа сеансов к каждому типу объектов. У сеанса определенного типа есть либо доступ только для чтения, либо доступ чтения/ записи к определенному типу объектов, либо к данному типу объектов доступа нет вообще.

Следует обратить внимание, что для создания или удаления объекта к нему нужен доступ чтения/ записи, например сеанс типа «Сеанс пользовательских функций только для чтения» не может создавать или удалять объект носителя.

Таблица 6 – Доступ сеансов различного типа к объектам различного типа

Тип объекта	Тип сеанса				
	Открытый сеанс только для чтения	Открытый сеанс чтение/запись	Пользовательский сеанс только для чтения	Пользовательский сеанс чтение/запись	Сеанс SO чтение/запись
Открытый объект сеанса	Чтение/ запись	Чтение/ запись	Чтение/ запись	Чтение/ запись	Чтение/ запись
Секретный объект сеанса			Чтение/ запись	Чтение/ запись	
Открытый объект носителя	Только чтение	Чтение/ запись	Только чтение	Чтение/ запись	Чтение/ запись
Секретный объект носителя			Только чтение	Чтение/ запись	

Как и говорилось ранее, доступ к определенному объекту сеанса, что показано в таблице 6, ограничен в пределах сеансов, которые принадлежат приложению, владеющему объектом (т. е. к приложению, которое создало данный объект).

6.7.3 События сеанса

События сеанса вызывают изменение состояния сеанса. В таблице 7 приводится описание состояний сеанса.

Таблица 7 – События сеанса

Событие	Происходит, когда …
Регистрация SO	SO прошел аутентификацию на носителе
Регистрация пользователя	Обычный пользователь прошел аутентификацию на носителе
Выход	Приложение осуществляет выход текущего пользователя из носителя (SO или обычный пользователь)
Закрытие сеанса	Приложение закрывает данный сеанс или все сеансы
Устройство удалено	Устройство, на котором работает носитель, было удалено из своего слота

При удалении устройства все сеансы всех приложений автоматически выходят из носителя. Более того, все сеансы, которые любые приложения имеют на данном устройстве, закрываются (такое поведение системы не было предусмотрено в Cryptoki версии 1.0) – приложение не может иметь сеанс с носителем, который отсутствует. В реальности Cryptoki не может постоянно следить за тем, есть ли носитель или его нет, и поэтому возможна ситуация, при которой отсутствие носителя не будет выявлено до тех пор, пока не начнут исполняться функции Cryptoki. Если до этого вставить носитель обратно в слот, Cryptoki может так и не выявить отсутствия носителя.

В Cryptoki все сеансы, которые приложение имеет с носителем, должны быть в одинаковом состоянии по регистрации/выходу (т. е. для определенного приложения и определенного носителя возможны варианты: все сеансы являются открытыми; все сеансы являются сеансами SO; все сеансы являются пользовательскими).

Когда сеанс приложения регистрируется на носителе, *все* сеансы, которые это приложение имеет с носителем, становятся зарегистрированными на нем, а когда приложение завершает сеанс выходом из носителя, все сеансы связи этого приложения с носителем закрываются. Аналогично, если, например, у приложения уже есть открытый пользовательский сеанс только для чтения с носителем, а затем оно открывает сеанс чтение/запись с этим носителем, то этот сеанс чтение/запись автоматически регистрируется на носителе.

Исходя из вышеизложенного, отдельное приложение не может одновременно иметь сеансы SO и пользовательские сеансы с отдельным носителем. Подразумевается также, что, если приложение имеет с носителем сеанс SO чтение/запись, то это приложение может не открывать с этим носителем сеанс только для чтения, так как сеансов SO только для чтения не существует. По той же причине, если приложение имеет открытым сеанс только для чтения, оно может не регистрировать на носителе какой-либо другой сеанс SO.

6.7.4 Идентификаторы сеанса и идентификаторы объекта

Идентификатор сеанса – это присвоенная интерфейсом Cryptoki величина, которая идентифицирует сеанс. Он во многом схож с идентификатором файла и указывается для функций, чтобы определить, с каким сеансом должна выполняться конкретная функция. Все потоки приложения имеют одинаковый доступ ко всем идентификаторам сеансов. Это означает, что все действия, которые можно выполнить с отдельным идентификатором файла через один поток, могут также выполняться с этим идентификатором файла и через любой другой поток того же приложения.

Cryptoki имеет также идентификаторы объектов, которые представляют собой идентификаторы, используемые для манипулирования объектами Cryptoki. Идентификаторы объектов похожи на идентификаторы сеансов с точки зрения того, что явность одного объекта через его идентификатор одинакова для всех потоков одного приложения. Конечно, сеансы только для чтения имеют к объектам носителя лишь доступ категории «только для чтения», тогда как у сеансов чтение/запись есть к этим объектам носителя доступ категории «чтение/запись».

В Cryptoki действительные идентификаторы сеансов и идентификаторы объектов всегда имеют значения, не равные нулю. Для удобства разработчиков Cryptoki определяет следующее символьное значение:

```
CK_INVALID_HANDLE
```

6.7.6 Возможности сеансов

Говоря весьма обобщенно, существует три широкие группы операций, которые могут выполняться с использованием открытого сеанса: административные операции (такие, как регистрация), операции по управлению объектами (такие, как создание или уничтожение объекта на носителе) и криптографические операции (такие, как вычисление списка сообщения). Криптографические операции иногда требуют запросить у Cryptoki API выполнение более чем одной функции. В общем, один сеанс может одновременно выполнять только одну операцию, по этой причине для одного приложения, возможно, нужно будет открыть несколько сеансов с одним носителем. Однако по соображениям эффективности один сеанс на некоторых носителях может одновременно выполнять пары операций следующих типов: получение списка сообщения и шифрование; расшифрование и получение списка сообщения; постановка электронной подписи или MAC и шифрование, а также расшифрование и верификация электронных подписей или MAC. Подробная информация о выполнении одновременных криптографических операций в одном сеансе представлена в 11.13.

Следствием того факта, что один сеанс может, как правило, одновременно выполнять только одну операцию, является то, что *приложение никогда не должно направлять делать из Cryptoki несколько одновременных вызовов функций, которые используют общий сеанс.* Cryptoki не определяет порядок действий на случай, если несколько потоков одного приложения попытаются таким образом

одновременно использовать общий сеанс. Это означает, что, если нескольким потокам одного приложения нужно использовать Cryptoki для получения доступа к конкретному носителю, возможно, каждому потоку нужно будет иметь собственный сеанс с этим носителем, если приложение не может каким-либо иным способом (например, при помощи какого-либо механизма блокировки) исключить одновременное использование одного сеанса несколькими потоками. Это верно для всех случаев, независимо от того, инициировалась ли библиотека Cryptoki в режиме, позволяющем обеспечивать безопасный многопоточный доступ к ней. Даже если безопасным будет одновременное получение доступа к библиотеке через несколько потоков, нельзя гарантировать безопасного использования *отдельного сеанса* для нескольких потоков одновременно.

6.7.7 Пример использования сеансов

Здесь приводится подробный пример того, как несколько приложений может использовать сеансы в библиотеке Cryptoki. Несмотря на достаточно сложный уровень детализации, настоятельно рекомендуем внимательно изучить данный пример, чтобы понять устройство идентификаторов сеансов и идентификаторов объектов.

Предупреждение. Данный пример намеренно *не* предназначен для того, чтобы показать, как несколько приложений *должны* использовать Cryptoki одновременно, а скорее, он предназначен для разъяснения того, какие варианты использования сеансов и объектов Cryptoki разрешены. Иными словами, вместо того, чтобы демонстрировать установленную методику, мы показываем, как эксплуатировать имеющиеся средства, выходя за рамки возможного.

Для нашего примера предположим, что приложения **A** и **B** используют библиотеку Cryptoki для получения доступа к одному носителю **T**. Оба приложения имеют по два работающих потока. **A** имеет потоки **A1** и **A2**, а **B** имеет потоки **B1** и **B2**. При последующих шагах считается, что ни на одном из них несколько потоков одного приложения не используют одновременно один и тот же сеанс и что события в примере происходят в указанном порядке, без взаимного перекрытия по времени.

1 **A1** и **B1** инициализируют библиотеку Cryptoki, вызывая **C_Initialize** (особенности функций Cryptoki объясняются в 10.2). Обратите внимание, что **C_Initialize** должна вызываться только один раз для каждого из приложений (а не, например, один раз для каждого потока).

2 **A1** открывает сеанс чтение/запись и получает для данного сеанса идентификатор сеанса 7. Так как это первый сеанс для приложения **A**, он является открытым сеансом.

3 **A2** открывает сеанс только чтения и получает идентификатор сеанса 4. Поскольку все существующие сеансы приложения **A** являются открытыми, сеанс 4 также будет открытым.

4 **A1** пытается зарегистрировать SO в сеансе 7. Попытка не удается, так как, если сеанс 7 станет сеансом SO, то сеанс 4 также станет таким сеансом, а сеансов SO только для чтения не существует. **A1** получает сообщение с кодом ошибки, указывающим на то, что существование сеанса только для чтения заблокировало данную попытку регистрации (CKR_SESSION_READ_ONLY_EXISTS).

5 **A2** регистрирует обычного пользователя в сеансе 7. Таким образом, сеанс 7 становится пользовательским сеансом чтения/записи, а сеанс 4 становится пользовательским сеансом только для чтения. Важно, что принадлежность **A1** и **A2** к одному приложению дает им равные права доступа ко всем сеансам и, следовательно, **A2** может выполнить данное действие.

6 **A2** открывает сеанс чтения/записи и получает идентификатор сеанса 9. Так как все существующие сеансы приложения **A** являются пользовательскими, то сеанс 9 также будет открытым пользовательским.

7 **A1** закрывает сеанс 9.

8 **B1** пытается выйти из сеанса 4. Попытка не удается, так как у **A** и **B** нет прав доступа к сеансам или объектам друг друга. **B1** получает сообщение об ошибке с указанием на отсутствие такого идентификатора сеанса (CKR_SESSION_HANDLE_INVALID).

9 **B2** пытается закрыть сеанс 4. Попытка не удается точно так же, как попытка **B1** выйти из сеанса 4 (т. е. **B2** получает сообщение с кодом ошибки CKR_SESSION_HANDLE_INVALID).

10 **B1** открывает сеанс чтения/записи и получает идентификатор сеанса 7. Для приложения **B** это первое появление идентификатора сеанса 7. Сеанс 7 приложения **A** и сеанс 7 приложения **B** – разные сеансы.

11 **B1** регистрирует SO в сеансе 7 [приложения **B**]. Это превращает сеанс 7 приложения **B** в сеанс чтения/записи SO, не влияя ни на один из сеансов приложения **A**.

12 **B2** пытается открыть сеанс только для чтения. Попытка не удается, так как у **B** уже есть открытый сеанс SO, а сеансов SO только для чтения не существует. **B1** получает сообщение об ошибке, указывающее на то, что наличие сеанса SO заблокировало данную попытку открыть сеанс только для чтения (CKR_SESSION_READ_WRITE_SO_EXISTS).

13 **A1** использует сеанс 7 [приложения **A**] для создания некоторого объекта сеанса **O1** и получает идентификатор объекта 7. Обратите внимание, что та или иная реализация Cryptoki может поддерживать, а может и не поддерживать отдельные участки для идентификаторов объектов и сеансов.

14 **B1** использует сеанс 7 [приложения **B**] для создания некоторого объекта носителя **O2** и получает идентификатор объекта 7. Как и в случае с идентификаторами сеансов, разные приложения не имеют прав доступа к идентификаторам объектов друг друга, и поэтому идентификатор объекта 7 приложения **B** и идентификатор объекта 7 приложения **A** – это совсем разные идентификаторы. Если **B1** – сеанс SO, он не может создавать секретные объекты, поэтому **O2** должен быть открытым объектом (если **B1** попытается создать секретный объект, попытка будет неудачной, при этом система выдаст сообщение с кодом ошибки CKR_USER_NOT_LOGGED_IN или CKR_TEMPLATE_INCONSISTENT).

15 **B2** использует сеанс 7 [приложения **B**] для выполнения некоторой операции по модификации объекта, привязанного к идентификатору объекта 7 [приложения **B**]. Это модифицирует объект **O2**.

16 **A1** использует сеанс 4 [приложения **A**] для выполнения операции поиска объекта в целях получения идентификатора для **O2**. Поиск выдает идентификатор объекта 1. Внимание! Идентификатор объекта 1 и идентификатор объекта 7 приложения **B** сейчас указывают на один и тот же объект.

17 **A1** пытается использовать сеанс 4 [приложения **A**], чтобы изменить объект, связанный с идентификатором объекта 1 [приложения **A**]. Попытка не удается, так как сеанс 4 приложения **A** – это сеанс только для чтения, и, следовательно, он не может изменить объект **O2**, который является объектом носителя. **A1** получает сообщение об ошибке, указывающее на то, что данный сеанс является сеансом только для чтения (CKR_SESSION_READ_ONLY).

18 **A1** использует сеанс 7 [приложения **A**], чтобы модифицировать объект, связанный с идентификатором объекта 1 [приложения **A**]. Так как сеанс 7 приложения **A** является сеансом чтения/записи, попытка удается, что приводит к изменению **O2**.

19 **B1** использует сеанс 7 [приложения **B**] для выполнения операции поиска объекта **O1.** Однако, поскольку **O1** является объектом сеанса, принадлежащего приложению **A**, поиск не удается.

20 **A2** использует сеанс 4 [приложения **A**] для выполнения некоторых операций по модификации объекта, связанного с идентификатором объекта 7 [приложения **A**]. Эта операция модифицирует объект **O1**.

21 **A2** использует сеанс 7 [приложения **A**] для уничтожения объекта, связанного с идентификатором объекта 1 [приложения **A**]. Это уничтожает объект **O2**.

22 **B1** пытается выполнить некоторые операции с объектом, привязанным к идентификатору объекта 7 [приложения **B**]. Попытка не удается, так как такого объекта больше не существует. **B1** получает сообщение об ошибке с указанием на то, что его идентификатор объекта недействителен (CKR_OBJECT_HANDLE_INVALID).

23 **A1** выходит из сеанса 4 [приложения **A**]. Так сеанс 4 [приложения **A**] становится открытым сеансом только для чтения, а сеанс 7 [приложения **A**] становится открытым сеансом чтения/записи.

24 **A1** закрывает сеанс 7 [приложения **A**]. Так уничтожается объект сеанса **O1**, который был создан сеансом 7 приложения **A**.

25 **A2** пытается использовать сеанс 4 [приложения **A**], чтобы выполнить некоторые операции с объектом, связанным с идентификатором объекта 7 [приложения **A**]. Попытка не удается, так как такой объект больше не существует. Выдается код ошибки CKR_OBJECT_HANDLE_INVALID.

26 **A2** выполняет вызов функции **C_CloseAllSessions**. Так закрывается сеанс 4 [приложения **A**]. В этот момент, если бы приложению **A пришлось** открывать новый сеанс, этот сеанс не был бы зарегистрирован (т. е. он был бы открытым сеансом).

27 **B2** закрывает сеанс 7 [приложения **B**]. В этот момент, если бы приложению **B** пришлось открывать новый сеанс, этот сеанс не был бы зарегистрирован.

28 Приложение **A** и приложение **B** производят вызовы функции **C_Finalize**, чтобы показать, что они закончили работу с библиотекой Cryptoki.

6.8 Повторная аутентификация (исключена из числа рекомендованных)

Примечание – Поддержка данного режима работы может быть представлена для обратной совместимости. Подробнее см. PKCS11 v 2.11.

6.9 Обзор функций

Cryptoki API состоит из некоторых функций, охватывающих управление носителем и слотом, управление объектом, а также криптографические функции. Эти функции представлены в таблице 8.

Таблица 8 – Краткое описание функций Cryptoki

Категория	Функция	Описание
Функции общего назначения	C_Initialize	Инициализирует Cryptoki
	C_Finalize	Очищает различные ресурсы, связанные с Cryptoki
	C_GetInfo	Получает общую информацию о Cryptoki
	C_GetFunctionList	Получает точки входа функций библиотеки Cryptoki
Функции управления слотами и носителями	C_GetSlotList	Получает список слотов системы
	C_GetSlotInfo	Получает информацию о конкретном слоте
	C_GetTokenInfo	Получает информацию о конкретном носителе
	C_WaitForSlotEvent	Ожидает события на слоте (вставка и удаление носителя и т. д.)
	C_GetMechanismList	Получает список механизмов, которые поддерживаются носителем
Функции управления слотами и носителями	C_GetMechanismInfo	Получает информацию о конкретном механизме
	C_InitToken	Инициализирует носитель
	C_InitPIN	Инициализирует PIN обычного пользователя
	C_SetPIN	Изменяет PIN текущего пользователя
Функции управления сеансом	C_OpenSession	Открывает соединение между приложением и конкретным носителем или настраивает обратный запрос приложения о присоединении носителя
	C_CloseSession	Закрывает сеанс
	C_CloseAllSessions	Закрывает все сеансы на носителе
	C_GetSessionInfo	Получает информацию о сеансе
	C_GetOperationState	Получает информацию о состоянии криптографических операций в ходе сеанса
	C_SetOperationState	Устанавливает состояние криптографических операций в ходе сеанса
	C_Login	Регистрируется на носителе
	C_Logout	Выходит из носителя
Функции управления объектами	C_CreateObject	Создает объект
	C_CopyObject	Создает копию объекта
	C_DestroyObject	Уничтожает объект
	C_GetObjectSize	Получает информацию о размере объекта в байтах
	C_GetAttributeValue	Получает информацию о величине атрибута объекта
	C_SetAttributeValue	Изменяет величину атрибута объекта
	C_FindObjectsInit	Инициализирует операцию поиска объекта
	C_FindObjects	Продолжает операцию поиска объекта
	C_FindObjectsFinal	Завершает операцию поиска объекта
Функции шифрования	C_EncryptInit	Инициализирует операцию шифрования
	C_Encrypt	Шифрует односторонние данные
	C_EncryptUpdate	Продолжает операцию многостороннего шифрования
	C_EncryptFinal	Завершает операцию многостороннего шифрования
Функции расшифрования	C_DecryptInit	Инициализирует операцию расшифрования
	C_Decrypt	Расшифровывает данные одностороннего шифрования
	C_DecryptUpdate	Продолжает операцию многостороннего расшифрования
	C_DecryptFinal	Завершает операцию многостороннего расшифрования

Окончание таблицы 8

Категория	Функция	Описание
Функции получения списка сообщений	C_DigestInit	Инициализирует операцию получения списка сообщений
	C_Digest	Получает список данных одностороннего шифрования
	C_DigestUpdate	Продолжает операцию многостороннего хэширования
	C_DigestKey	Получает список ключей
	C_DigestFinal	Завершает многостороннюю операцию получения списка
Функции подписи и MAC	C_SignInit	Инициализирует операцию постановки электронной подписи
	C_Sign	Подписывает односторонние данные
	C_SignUpdate	Продолжает операцию многосторонней подписи
	C_SignFinal	Завершает операцию многосторонней подписи
	C_SignRecoverInit	Инициализирует операцию подписи, где данные могут быть восстановлены из подписи
	C_SignRecover	Подписывает односторонние данные при осуществлении операции, где данные могут быть восстановлены из подписи
Функции верификации подписей и имитовставок	C_VerifyInit	Инициализирует операцию верификации
	C_Verify	Производит верификацию односторонней электронной подписи
	C_VerifyUpdate	Продолжает операцию верификации многосторонней электронной подписи
	C_VerifyFinal	Завершает операцию верификации данных многостороннего шифрования
	C_VerifyRecoverInit	Инициализирует операцию верификации, когда данные могут быть извлечены из подписи
	C_VerifyRecover	Производит верификацию подписи при осуществлении операции, где данные могут быть восстановлены из подписи
Криптографические функции двойного назначения	C_DigestEncryptUpdate	Продолжает одновременные многосторонние операции получения списка и шифрования
	C_DecryptDigestUpdate	Продолжает одновременные многосторонние операции получения списка и расшифрования
	C_SignEncryptUpdate	Продолжает одновременные многосторонние операции постановки эклектроной подписи и шифрования
	C_DecryptVerifyUpdate	Продолжает одновременные многосторонние операции расшифрования и верификации
Функции распределения ключей	C_GenerateKey	Генерирует секретный ключ
	C_GenerateKeyPair	Генерирует пару открытый ключ/секретный ключ
	C_WrapKey	Свертывает ключ
	C_UnwrapKey	Разворачивает ключ
	C_DeriveKey	Выводит ключ из базового ключа
Функции генерации случайных чисел	C_SeedRandom	Подмешивает дополнительный исходный материал в генератор случайных чисел
	C_GenerateRandom	Генерирует случайные данные
Функции управления параллельными функциями	C_GetFunctionStatus	Унаследованная функция, которая всегда выдает CKR_FUNCTION_NOT_PARALLEL
	C_CancelFunction	Унаследованная функция, которая всегда выдает CKR_FUNCTION_NOT_PARALLEL
Функция обратного вызова		Предоставляемая приложением функция обработки уведомлений от Cryptoki

7 Анализ безопасности

Являясь интерфейсом для криптографических устройств, Cryptoki составляет основу обеспечения безопасности в компьютере или в системе связи. Ниже приводится описание специфических свойств интерфейса, которые обеспечивают такую безопасность:

1. Доступ к секретным объектам на носителе, а также, возможно, к криптографическим функциям и/или сертификатам на носителе, требует ввода PIN. Таким образом, просто наличия криптографического устройства, применяющего носитель, может быть недостаточно для его использования, возможно, также понадобится PIN.

2. Дополнительная защита личных/секретных и секретных ключей может осуществляться путем пометки их как «уязвимых» или «неизвлекаемых». Уязвимые ключи нельзя просмотреть в открытом тексте из носителя, а неизвлекаемые ключи нельзя просмотреть из носителя даже при шифровании (хотя они все же могут использоваться как ключи).

Ожидается, что получить доступ к секретным, уязвимым или неизвлекаемым объектам при помощи любых средств, кроме Cryptoki (например, другие программные интерфейсы, реинжиниринг устройства), будет сложно.

Если устройство не имеет оболочки защиты от несанкционированного доступа или защищенной памяти для хранения секретных и уязвимых объектов, то это устройство может шифровать объекты мастер-ключом, который, возможно, будет выведен из пользовательского PIN. Однако конкретный механизм защиты секретных объектов зависит от реализации устройства.

Наличие этих свойств должно сделать возможным построение приложений таким образом, чтобы носитель мог обеспечивать достаточную безопасность для объектов, которые управляются приложением.

Криптография – лишь один из элементов безопасности, а носитель – лишь один из компонентов системы. Тогда как сам по себе носитель может быть безопасным, нужно учитывать еще и безопасность операционной системы, через которую приложение соединяется с ним, особенно в связи с тем, что через эту операционную систему может передаваться PIN. Данное обстоятельство может предоставить возможность инородному приложению на этой операционной системе легко получить PIN; также возможен вариант, при котором другие устройства, контролирующие каналы связи с криптографическим устройством, смогут получить информацию о значении PIN. Инородные приложения и устройства могут также изменять команды, которые отправляются криптографическому устройству, в целях получения тех сервисов, которые не запрашивались приложением.

Важно иметь уверенность в том, что система является безопасной от таких видов атак. Для разрешения данной проблемы вполне возможно применение Cryptoki; например, носитель может принимать участие в загрузке системы.

Необходимо иметь ввиду то, что ни одна из атак, описание которых только что приводилось, не может обеспечить несанкционированный доступ к ключам, которые помечены как «уязвимые», так как ключ, который является уязвимым, всегда останется уязвимым. Точно так же, ключ, который является неизвлекаемым, не может быть изменен и представлен как извлекаемый.

Для приложения важно также иметь уверенность в том, что носитель является в определенном смысле «законным» (по разным причинам), включая ограничения по экспорту и базовые вопросы безопасности. Данная тема не рассматривается в настоящем стандарте, однако решение такого вопроса может заключаться в распространении носителей со встроенной и сертифицированной парой секретный ключ/открытый ключ, при помощи которой может удостоверяться подлинность носителя. Сертификат будет подписываться представителем полномочного государственного органа (вероятно, органа, который удостоверяет, что носитель является законным), чей открытый ключ известен приложению. Приложение будет проверять сертификат и испытывать носитель на подлинность путем подписания сообщения, которое не соответствует по времени данным его встроенного секретного ключа.

После того, как обычный пользователь прошел аутентификацию в носителе, Cryptoki не ограничивает данного пользователя в количестве криптографических операций, которые тот может выполнять; этот пользователь может выполнять любые операции, которые поддерживаются носителем. В некоторых носителях, возможно, и не нужна будет никакая аутентификация, для того чтобы использовать его криптографические функции.

8 Инструкции C или C++, зависящие от платформы и компилятора

Существует большая совокупность типов данных, связанных с Cryptoki, которые определяются в заголовочных файлах Cryptoki. Некоторые вопросы, связанные с этими типами данных, относящиеся к упаковке и указателям, зависимы от платформы и компилятора, поэтому такие вопросы разрешаются на основе платформа − платформа (или компилятор − компилятор) вне заголовочных файлов

Criptoki средствами предпроцессорных инструкций.

Это означает, что при написании программного кода C или C++ некоторые инструкции предпроцессора должны задаваться до включения в код заголовочных файлов Cryptoki. Описание этих инструкций приводится в оставшейся части раздела 8.

8.1 Упаковка структуры

Структуры данных Cryptoki упакованы таким образом, чтобы занимать как можно меньше места. В частности, на платформах Win32 и Win16 структуры данных Cryptoki должны быть упакованы с 1-байтным выравниванием. В среде UNIX изменение байтового выравнивания структур может быть, а может и не быть обязательным (или даже возможным).

8.2 Макросы, связанные с указателем

В связи с тем, что различные платформы и компиляторы используют различные способы работы с различными типами указателей, для работы с Cryptoki нужно, чтобы следующие 6 макросов были выставлены вне системы Cryptoki:

♦ **CK_PTR**

`CK_PTR` – это «косвенная строка», которую определенный компилятор или определенная платформа использует для создания указателя на объект. Она используется следующим образом:

```
typedef CK_BYTE CK_PTR CK_BYTE_PTR;
```

♦ **CK_DEFINE_FUNCTION**

Макрос `CK_DEFINE_FUNCTION(returnType, name)`, когда после него следуют круглые скобки, содержащие список аргументов и определение функции, определяет функцию Cryptoki API в библиотеке Cryptoki. `returnType` – это тип выдаваемых функцией данных, а `name` – это ее наименование. Данный макрос используется следующим образом:

```
CK_DEFINE_FUNCTION(CK_RV, C_Initialize)(
  CK_VOID_PTR pReserved
)
{
 ...
}
```

♦ **CK_DECLARE_FUNCTION**

Макрос `CK_DEFINE_FUNCTION(returnType, name)`, когда после него следуют круглые скобки, содержащие список аргументов, и точка с запятой, объявляет функцию Cryptoki API в библиотеке Cryptoki. `returnType` – это тип выдаваемых функцией данных, а `name` – это ее наименование. Данный макрос используется следующим образом:

```
CK_DECLARE_FUNCTION(CK_RV, C_Initialize)(
  CK_VOID_PTR pReserved
);
```

♦ **CK_DECLARE_FUNCTION_POINTER**

Макрос `CK_DEFINE_FUNCTION(returnType, name)`, когда после него следуют круглые скобки, содержащие список аргументов, и точка с запятой, объявляет функцию Cryptoki API в библиотеке Cryptoki. `returnType` – это тип выдаваемых функцией данных, а `name` – это ее наименование. Он может использоваться одним из двух указанных ниже способов для задания переменной указателя функции, `myC_Initialize`, которая может указывать на функцию **C_Initialize** в библиотеке Cryptoki (ни один из последующих фрагментов программного кода фактически не *присваивает* значения переменной `myC_Initialize`):

```
CK_DECLARE_FUNCTION_POINTER(CK_RV, myC_Initialize)(
  CK_VOID_PTR pReserved
);
```

или

```
typedef CK_DECLARE_FUNCTION_POINTER(CK_RV, myC_InitializeType)(
  CK_VOID_PTR pReserved
);
myC_InitializeType myC_Initialize;
```

♦ **CK_CALLBACK_FUNCTION**

Макрос `CK_DEFINE_FUNCTION(returnType, name)`, когда после него следуют круглые скобки, содержащие список аргументов, и точка с запятой, объявляет переменную или тип, который является указателем на функцию обратного вызова приложения, которая может использоваться функцией Cryptoki API в библиотеке Cryptoki. `returnType` – это тип выдаваемых функцией данных, а `name` – это ее наименование. Он может использоваться одним из двух указанных ниже способов для задания переменной указателя функции, `myCallback`, которая может указывать на обратный вызов приложения, использующего аргументы args, и выдает **CK_RV** (ни один из последующих фрагментов программного кода фактически не *присваивает* значения переменной `myCallback`):

```
CK_CALLBACK_FUNCTION(CK_RV, myCallback)(args);
```

или

```
typedef CK_CALLBACK_FUNCTION(CK_RV, myCallbackType)(args);
myCallbackType myCallback;
```

♦ **NULL_PTR**

`NULL_PTR` – это значение пустого указателя NULL. В любой среде ANSI C (а также во многих других) `NULL_PTR` должен задаваться просто как 0.

8.3 Пример кода, зависящего от платформы и компилятора

8.3.1 Win32

Разработчики, использующие Microsoft Developer Studio 5.0 для создания программного кода C или C++, который реализует или использует библиотеку Win32 Cryptoki.dll, могут выдавать следующие инструкции до включения в программный код каких-либо заголовочных файлов Cryptoki:

```
#pragma pack(push, cryptoki, 1)

#define CK_IMPORT_SPEC __declspec(dllimport)

/* Define CRYPTOKI_EXPORTS during the build of cryptoki
 * libraries. Do not define it in applications.
 */
#ifdef CRYPTOKI_EXPORTS
#define CK_EXPORT_SPEC __declspec(dllexport)
#else
#define CK_EXPORT_SPEC CK_IMPORT_SPEC
#endif

/* Ensures the calling convention for Win32 builds */
#define CK_CALL_SPEC __cdecl

#define CK_PTR *

#define CK_DEFINE_FUNCTION(returnType, name) \
  returnType CK_EXPORT_SPEC CK_CALL_SPEC name

#define CK_DECLARE_FUNCTION(returnType, name) \
  returnType CK_EXPORT_SPEC CK_CALL_SPEC name

#define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
  returnType CK_IMPORT_SPEC (CK_CALL_SPEC CK_PTR name)

#define CK_CALLBACK_FUNCTION(returnType, name) \
```

```
  returnType (CK_CALL_SPEC CK_PTR name)

#ifndef NULL_PTR
#define NULL_PTR 0
#endif
```

Поэтому соглашение о вызовах (*определение порядка размещения в стеке и извлечения из стека параметров, передаваемых при вызове функций и возврате в вызывающую программу*) для всех функций C_xxx должно соответствовать «cdecl», когда параметры функции передаются справа налево, а инициатор запроса удаляет параметры из стека при возврате вызова.

После включения в программный код любых заголовочных файлов Cryptoki они могут выдавать следующие инструкции для задания упаковки структуры в ее первоначальном значении:

```
#pragma pack(pop, cryptoki)
```

8.3.2 Win16

Разработчики, использующие версии Microsoft Developer Studio ниже 5.0 для создания программного кода C или C++, который реализует или использует библиотеку Win32 Cryptoki.dll, могут выдавать следующие инструкции до включения в программный код каких-либо заголовочных файлов Cryptoki:

```
#pragma pack(1)

#define CK_PTR far *

#define CK_DEFINE_FUNCTION(returnType, name) \
  returnType __export _far _pascal name

#define CK_DECLARE_FUNCTION(returnType, name) \
  returnType __export _far _pascal name

#define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
  returnType __export _far _pascal (* name)

#define CK_CALLBACK_FUNCTION(returnType, name) \
  returnType _far _pascal (* name)

#ifndef NULL_PTR
#define NULL_PTR 0
#endif
```

8.3.3 Общий UNIX

Разработчики, использующие общий UNIX, могут выдавать следующие инструкции до включения в программный код каких-либо заголовочных файлов Cryptoki:

```
#define CK_PTR *

#define CK_DEFINE_FUNCTION(returnType, name) \
  returnType name

#define CK_DECLARE_FUNCTION(returnType, name) \
  returnType name

#define CK_DECLARE_FUNCTION_POINTER(returnType, name) \
  returnType (* name)

#define CK_CALLBACK_FUNCTION(returnType, name) \
  returnType (* name)

#ifndef NULL_PTR
#define NULL_PTR 0
#endif
```

9 Общие типы данных

В следующих подразделах приводится описание общих типов данных Cryptoki. Не приводится описание типов данных, которые используются для хранения параметров различных механизмов, а также указателей на эти параметры; описание этих типов данных вместе с информацией по самим механизмам приводится в разделе 12.

Исходный файл C или C++ в библиотеке или приложении Cryptoki может описывать все типы данных (типы, описание которых приводится здесь, а также типы, которые специально используются для определенных параметров механизма) путем включения файла Cryptoki верхнего уровня `pkcs11.h`. В свою очередь `pkcs11.h` включает другие файлы Cryptoki, `pkcs11t.h` и `pkcs11f.h`. Исходный файл может также включать просто `pkcs11t.h` вместо `pkcs11.h`); таким образом задаются многие (но не все) из типов, указанных здесь.

При включении любого из этих заголовочных файлов исходный файл должен задавать инструкции предпроцессора, указанные в разделе 8.

9.1 Общая информация

Cryptoki представляет общую информацию следующими типами данных:

♦ **CK_VERSION; CK_VERSION_PTR**

CK_VERSION – это структура, описывающая версию интерфейса Cryptoki, библиотеки Cryptoki или реализации SSL, или же программную либо аппаратную версию слота или носителя. Данная структура задается следующим образом:

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

Поля данной структуры имеют значения:

major основной номер версии (целая часть номера версии);

minor дополнительный номер версии (дробная часть номера версии).

Пример – Для версии 1.0 major = 1, a minor = 0. Для версии 2.10 major = 2, a minor = 10. В таблице 9 приводится список значений для основных и дополнительных номеров версий официально опубликованных реализаций Cryptoki.

Таблица 9 – Значения основных и дополнительных номеров версий, опубликованных спецификаций Cryptoki

Версия	major	minor
1.0	0x01	0x00
2.01	0x02	0x01
2.10	0x02	0x0a
2.11	0x02	0x0b
2.20	0x02	0x14

Модификации стандарта Cryptoki, различающиеся лишь дополнительными номерами, всегда совместимы с более ранними версиями того же основного номера.

CK_VERSION_PTR является указателем на **CK_VERSION**.

♦ **CK_INFO; CK_INFO_PTR**

CK_INFO предоставляет общую информацию о Cryptoki. Данная структура описывается следующим образом:

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

Поля данной структуры имеют значения:

cryptokiVersion номер версии интерфейса Cryptoki для совместимости с последующими модификациями этого интерфейса;

manufacturerID идентификационный номер поставщика библиотеки Cryptoki. Перед ним и после него должны стоять символы пробела (‘ ‘). Данное поле *не* должно завершаться нулем;

flags флаги, зарезервированные для последующих версий. Для данной версии это должен быть ноль;

libraryDescription описание символьной строки библиотеки. Перед ним и после него должны стоять символы пробела (‘ ‘). Данное поле *не* должно завершаться нулем;

libraryVersion номер версии библиотеки Cryptoki.

У библиотек, составленных для данного документа, значение *cryptokiVersion* должно совпадать с версией данного документа; значение *libraryVersion* – это номер версии самого ПО библиотеки.

CK_ INFO _PTR – это указатель на **CK_ INFO**.

♦ **CK_NOTIFICATION**

CK_NOTIFICATION хранит типы уведомлений, которые Cryptoki предоставляет приложению. Данная структура описывается следующим образом:

```
typedef CK_ULONG CK_NOTIFICATION;
```

Для данной версии Cryptoki определены следующие типы уведомлений:

```
CKN_SURRENDER
```

Эти уведомления имеют значения:

CKN_SURRENDER Cryptoki отказывается от выполнения функции, которая выполняется в ходе сеанса, чтобы приложение могло выполнять другие операции. После выполнения любых желаемых операций приложение должно указать Cryptoki, продолжать или отменять выполнение функции (см. 11.17.1).

9.2 Типы слота и носителя

Cryptoki представляет информацию о слоте и носителе при помощи данных следующих типов:

♦ **CK_SLOT_ID; CK_SLOT_ID_PTR**

CK_SLOT_ID – это значение, назначенное интерфейсом Cryptoki и определяющее слот. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_SLOT_ID;
```

C_GetSlotList выдает список значений **CK_SLOT_ID**. Изначально *любое* значение **CK_SLOT_ID** может быть действительным идентификатором слота (в частности, в системе может быть слот, заданный значением 0). Однако не обязательно иметь такой слот.

CK_SLOT_ID_PTR – это указатель на **CK_SLOT_ID**.

♦ **CK_SLOT_INFO; CK_SLOT_INFO_PTR**

CK_SLOT_INFO предоставляет информацию о слоте. Данная структура задается следующим образом:

```
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 данной структуры имеют значения:

slotDescription описание символьной строки слота. Перед ним и после него должны стоять символы пробела (‘ ‘). Данное поле *не* должно завершаться нулем;

manufacturerID идентификационный номер поставщика слота. Перед ним и после него должны

стоять символы пробела (‘ ‘). Данное поле *не* должно завершаться нулем;
flags флаги, которые указывают возможности слота. Флаги определяются ниже;
hardwareVersion номер версии аппаратного решения слота;
firmwareVersion номер версии программного обеспечения слота.

В таблице 10 описывается поле *флагов*.

Таблица 10 – Информационные флаги слота

Флаг	Маска	Значение
CKF_TOKEN_PRESENT	0x00000001	Истинно, если в слоте присутствует носитель (например, в считывателе есть устройство)
CKF_REMOVABLE_DEVICE	0x00000002	Истинно, если считыватель поддерживает съемные устройства
CKF_HW_SLOT	0x00000004	Истинно, если слот является аппаратным слотом, а не программным слотом, реализующим программный носитель

Для конкретного слота значение флага **CKF_REMOVABLE_DEVICE** *никогда не меняется*. Кроме того, если этот флаг не выставлен для конкретного слота, для этого слота *всегда* выставлен флаг **CKF_TOKEN_PRESENT**. Это означает, что если слот не поддерживает съемное устройство, в нем всегда находится носитель.

CK_SLOT_INFO_PTR – указатель на **CK_SLOT_INFO**.

♦ **CK_TOKEN_INFO; CK_TOKEN_INFO_PTR**

CK_SLOT_INFO предоставляет информацию о носителе. Данная структура задается следующим образом:

```
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 данной структуры имеют следующие значения:

label метка, которая задается приложением и присваивается во время инициализации носителя. Перед ней и после нее должны стоять символы пробела (‘ ‘). Данное поле *не* должно завершаться нулем;

manufacturerID идентификационный номер поставщика устройства. Приводится в окружении пустых символов (‘ ‘). Перед ним и после него должны стоять символы пробела (‘ ‘). Данное поле *не* должно завершаться нулем;

model модель устройства. Перед данным полем и после него должны стоять символы пробела (‘ ‘). Данное поле *не* должно завершаться нулем;

serialNumber символьная строка с серийным номером устройства. Перед данным полем и после него должны стоять символы пробела (‘ ‘). Данное поле *не* должно завершаться нулем;

flags флаги, указывающие возможности и состояние устройства, в соответ-

ствии с определениями, которые приводятся ниже;

ulMaxSessionCount максимальное количество сеансов, которые могут одновременно быть открыты с носителем одним приложением (см. примечание ниже);

ulSessionCount количество сеансов, которые на данный момент открыты между приложением и носителем (см. примечание ниже);

ulMaxRwSessionCount максимальное количество сеансов чтения/записи, которые могут одновременно быть открыты с носителем одним приложением (см. примечание ниже);

ulRwSessionCount количество сеансов чтения/записи, которые на данный момент открыты между приложением и носителем (см. примечание ниже);

ulMaxPinLen максимальная длина значения PIN в байтах;

ulMinPinLen минимальная длина значения PIN в байтах;

ulTotalPublicMemory общий объем в байтах участков памяти на носителе, которые могут использоваться для хранения открытых объектов (см. примечание ниже)

ulFreePublicMemory объем в байтах участков свободной (неиспользуемой) памяти на носителе для открытых объектов (см. примечание ниже);

ulTotalPrivateMemory общий объем в байтах участков памяти на носителе, которые могут использоваться для хранения секретных объектов (см. примечание ниже)

ulFreePrivateMemory объем в байтах участков свободной (неиспользуемой) памяти на носителе для секретных объектов (см. примечание ниже);

hardwareVersion номер версии аппаратного решения;

firmwareVersion номер версии программного обеспечения;

utcTime текущее время в виде символьной строки в формате YYYYMMDDhhmmssxx (4 символа для года; по два символа на месяц, день, час, минуты и секунды; два дополнительно зарезервированных символа ‘0’). Значение данного поля имеет смысл только для носителей, снабженных часами, как обозначено в информационных флагах носителя (см. ниже).

В таблице 11 описаны поля *флагов*.

Таблица 11 – Информационные флаги носителя

Флаг	Маска	Значение
CKF_RNG	0x00000001	Истинно, если носитель имеет собственный генератор случайных чисел
CKF_WRITE_PROTECTED	0x00000002	Истинно, если носитель защищен от записи (см. ниже)
CKF_LOGIN_REQUIRED	0x00000004	Истинно, если есть такие криптографические функции, для выполнения которых пользователь должен быть зарегистрирован
CKF_USER_PIN_INITIALIZED	0x00000008	Истинно, если инициализирован PIN обычного пользователя
CKF_RESTORE_KEY_NOT_NEEDED	0x00000020	Верно, если успешное сохранение состояния криптографических операций сеанса *всегда* сохраняет все ключи, необходимые для восстановления состояния сеанса
CKF_CLOCK_ON_TOKEN	0x00000040	Истинно, если в носителе есть собственные встроенные часы
CKF_PROTECTED_AUTHENTICATION_PATH	0x00000100	Истинно, если в носителе есть «защищенный путь аутентификации», при помощи которого пользователь может зарегистрироваться в носителе без необходимости передавать PIN через библиотеку Cryptoki
CKF_DUAL_CRYPTO_OPERATIONS	0x00000200	Истинно, если один сеанс с носителем может выполнять двойные криптографические операции (см. 11.13)

Окончание таблицы 11

Флаг	Маска	Значение
CKF_TOKEN_INITIALIZED	0x00000400	Истинно, если носитель инициализировался с использованием C_InitializeToken или эквивалентного механизма, который не охватывается настоящим стандартом. Вызов C_InitializeToken, когда выставлен этот флаг, приведет к реинициализации носителя
CKF_SECONDARY_AUTHENTICATION	0x00000800	Истинно, если носитель поддерживает повторную аутентификацию объектов секретного ключа. (исключен из числа рекомендованных; в новых реализациях этот флаг выставляться НЕ ДОЛЖЕН)
CKF_USER_PIN_COUNT_LOW	0x00010000	Истинно, если введен неверный PIN пользователя по крайней мере один раз с момента последней успешной аутентификации
CKF_USER_PIN_FINAL_TRY	0x00020000	Истинно, если ввод неверного пользовательского PIN приведет к блокировке носителя
CKF_USER_PIN_LOCKED	0x00040000	Истинно, если пользовательский PIN заблокирован. Регистрация пользователя на носителе невозможна
CKF_USER_PIN_TO_BE_CHANGED	0x00080000	Истинно, если значение пользовательского PIN является значением по умолчанию, которое выставляется при инициализации либо при производстве носителя, или действие PIN прекращено
CKF_SO_PIN_COUNT_LOW	0x00100000	Истинно, если ответственным по безопасности введен неверный PIN по меньшей мере один раз с момента последней успешной аутентификации
CKF_SO_PIN_FINAL_TRY	0x00200000	Истинно, если ввод ответственным за безопасность неверного PIN приведет к блокировке носителя
CKF_SO_PIN_LOCKED	0x00400000	Истинно, если PIN ответственного за безопасность заблокирован. Регистрация пользователя на носителе невозможна
CKF_SO_PIN_TO_BE_CHANGED	0x00800000	Истинно, если значение PIN ответственного за безопасность является значением по умолчанию, которое выставляется при инициализации либо производстве носителя, или действие PIN прекращено

В Cryptoki точно не определено, что именно означает флаг **CKF_WRITE_PROTECTED**. Приложение может и не быть способным выполнять определенные операции на носителе, защищенном от записи; эти действия, помимо прочего, могут включать любые из следующих:

– создание/изменение/удаление любого объекта на носителе;

– создание/изменение/удаление объекта носителя на носителе;

– изменение PIN ответственного за безопасность (SO);

– изменение PIN обычного пользователя.

Носитель может менять значение флага **CKF_WRITE_PROTECTED** в зависимости от состояния сеанса, чтобы применить свою политику управления объектами. Например, носитель может выставить флаг **CKF_WRITE_PROTECTED**, если состояние сеанса не является состоянием «сеанс SO чтения/ записи» или «пользовательский сеанс чтения/записи», чтобы применить свою политику, которая не разрешает создавать, изменять или удалять объекты как секретные, так и открытые, если пользователь не произвел успешный вызов C_Login.

Флаги **CKF_USER_PIN_COUNT_LOW**, **CKF_USER_PIN_COUNT_LOW**, **CKF_USER_PIN_FINAL_TRY** и **CKF_SO_PIN_FINAL_TRY** всегда могут быть выставлены на «ложно», если носитель не поддерживает такую функциональность или не раскрывает информацию, следуя своей политике безопасности.

Флаги **CKF_USER_PIN_TO_BE_CHANGED** и **CKF_SO_PIN_TO_BE_CHANGED** всегда могут быть выставлены на «ложно», если носитель не поддерживает такую функциональность. Если PIN выставлен на значение по умолчанию или его действие прекратилось, соответствующий флаг − **CKF_USER_PIN_TO_BE_CHANGED** или **CKF_SO_PIN_TO_BE_CHANGED** будет выставлен на «истинно». Если на «истинно» выставлен один из этих двух флагов, попытка входа с соответствующим PIN будет удачной, но вызвана может быть только функция C_SetPIN. Вызов любой другой функции, которая требует
регистрации пользователя, будет приводить к выдаче сообщения CKR_PIN_EXPIRED, пока не будет успешно вызвана C_SetPIN.

Примечание – Поля *ulMaxSessionCount*, *ulSessionCount*, *ulMaxRwSessionCount*, *ulRwSessionCount*, *ulTotalPublicMemory*, *ulFreePublicMemory*, *ulTotalPrivateMemory* и *ulFreePrivateMemory* могут иметь специальное значение CK_UNAVAILABLE_INFORMATION, которое означает, что носитель и/или библиотека не может или не желает предоставлять данную информацию. Кроме того, поля *ulMaxSessionCount* и *ulMaxRwSessionCount* могут иметь специальное значение CK_EFFECTIVELY_INFINITE, которое означает, что не существует фактического ограничения по количеству сеансов (соответственно, сеансов чтения/записи), которые приложение может открыть с носителем.

Важно проверить эти поля на наличие в них этих специальных значений. Это особенно верно для CK_EFFECTIVELY_INFINITE, так как приложение, видя данное значение в полях *ulMaxSessionCount* или *ulMaxRwSessionCount*, напротив, посчитает, что оно не может открыть *ни одного* сеанса с носителем, а это абсолютно неверно.

Результат всего вышеизложенного заключается в том, что правильная интерпретация, например поля *ulMaxSessionCount*, – это что-то похожее на:

```
CK_TOKEN_INFO info;

.
.
if ((CK_LONG) info.ulMaxSessionCount
  == CK_UNAVAILABLE_INFORMATION) {
  /* Token refuses to give value of ulMaxSessionCount */
  .
  .
} else if (info.ulMaxSessionCount == CK_EFFECTIVELY_INFINITE) {
  /* Application can open as many sessions as it wants */
  .
  .
} else {
  /* ulMaxSessionCount really does contain what it should */
  .
  .
}
```

CK_TOKEN_INFO_PTR – это указатель на **CK_TOKEN_INFO**.

9.3 Типы сеанса

Cryptoki представляет информацию о сеансе с использованием следующих типов:

- **CK_SESSION_HANDLE; CK_SESSION_HANDLE_PTR**

CK_SLOT_ID – это значение, которое присваивается Cryptoki и задает сеанс. Оно задается следующим образом:

```
typedef CK_ULONG CK_SESSION_HANDLE;
```

В Cryptoki действительные идентификаторы сеансов всегда имеют значения, не равные нулю. Для удобства разработчиков Cryptoki определяет следующее символьное значение:

```
CK_INVALID_HANDLE
```

CK_SESSION_HANDLE_PTR – это указатель на **CK_SESSION_HANDLE**.

♦ **CK_USER_TYPE**

CK_USER_TYPE содержит типы пользователей Cryptoki, описание которых приводилось в 6.5, а кроме того, специализированный, определяемый контекстом тип, описание которого приводится в 10.9. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_USER_TYPE;
```

Для данной версии Cryptoki определены следующие типы пользователей:

```
CKU_SO
CKU_USER
CKU_CONTEXT_SPECIFIC
```

♦ **CK_STATE**

CK_STATE содержит состояние сеанса, как это описано в 6.7.1 и 6.7.2. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_STATE;
```

Для данной версии Cryptoki определены следующие состояния сеанса:

```
CKS_RO_PUBLIC_SESSION
CKS_RO_USER_FUNCTIONS
CKS_RW_PUBLIC_SESSION
CKS_RW_USER_FUNCTIONS
CKS_RW_SO_FUNCTIONS
```

♦ **CK_SESSION_INFO; CK_SESSION_INFO_PTR**

CK_SESSION_INFO предоставляет информацию о сеансе. Данная структура задается следующим образом:

```
typedef struct CK_SESSION_INFO {
 CK_SLOT_ID slotID;
 CK_STATE state;
 CK_FLAGS flags;
 CK_ULONG ulDeviceError;
} CK_SESSION_INFO;
```

Поля данной структуры имеют следующие значения:

slotID ID слота, соединяющегося с носителем;
state состояние сеанса;
flags флаги, которые определяют тип сеанса; ниже приводится описание таких флагов;
ulDeviceError код ошибки, который определяется криптографическим устройством. Используется для указания на ошибки, не охватываемые действием Cryptoki.

В таблица 12 определены поля *флагов*.

Таблица 12 – Флаги информации о сеансе

Флаг	Маска	Значение
CKF_RW_SESSION	0x00000002	Истинно, если это сеанс чтения/записи; ложно, если это сеанс только для чтения
CKF_SERIAL_SESSION	0x00000004	Этот флаг предназначен для обратной совместимости и должен всегда быть выставлен на истинно

CK_SESSION_INFO_PTR – это указатель на **CK_SESSION_INFO**.

9.4 Типы объекта

Cryptoki представляет информацию об объектах с использованием следующих типов:

♦ **CK_OBJECT_HANDLE; CK_OBJECT_HANDLE_PTR**

CK_OBJECT_HANDLE – идентификатор объекта, соответствующий носителю. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_OBJECT_HANDLE;
```

Когда приложение создает или находит объект на носителе, Cryptoki присваивает ему идентификатор объекта, который сеанс приложения будет использовать для доступа к нему. У определенного объекта на носителе не обязательно есть идентификатор, который зафиксирован за ним на все время существования объекта, однако, если определенный сеанс может использовать определенный идентификатор для доступа к определенному объекту, то сеанс будет и в дальнейшем иметь возможность использовать этот идентификатор для доступа к данному объекту на протяжении того времени, пока сам этот сеанс продолжает существовать, а данный объект продолжает быть доступным для данного сеанса.

В Cryptoki действительные идентификаторы объектов всегда имеют значения, не равные нулю. Для удобства разработчиков Cryptoki определяет следующее символьное значение:

```
CK_INVALID_HANDLE
```

CK_OBJECT_HANDLE_PTR – указатель на **CK_OBJECT_HANDLE**.

♦ **CK_OBJECT_CLASS; CK_OBJECT_CLASS_PTR**

CK_OBJECT_CLASS – это значение, определяющее классы (или типы) объектов, которые распознает Cryptoki. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_OBJECT_CLASS;
```

Классы объектов описываются вместе с объектами, которые используют их. Тип указывается в объекте через атрибут объекта CKA_CLASS.

Могут также указываться значения для данного типа, определяемые поставщиком.

```
CKO_VENDOR_ DEFINED
```

Классы объектов **CKO_VENDOR_DEFINED**, а также те, что указаны выше, постоянно зарезервированы для поставщиков носителя. В целях совместимости поставщики должны регистрировать свои классы объектов через процесс PKCS.

CK_OBJECT_CLASS_PTR – это указатель на **CK_OBJECT_CLASS**.

♦ **CK_HW_FEATURE_TYPE**

CK_HW_FEATURE_TYPE – это значение, которое указывает на тип устройства по аппаратным характеристикам. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_HW_FEATURE_TYPE;
```

Типы устройства по аппаратным характеристикам описываются вместе с объектами, которые используют их. Тип указывается в объекте через атрибут объекта CKA_HW_FEATURE_TYPE.

Могут также указываться значения для данного типа, определяемые поставщиком.

```
CKH_VENDOR_DEFINED
```

Типы свойств **CKH_VENDOR_DEFINED**, а также те, что указаны выше, постоянно зарезервированы для поставщиков носителя. В целях функциональной совместимости поставщики должны регистрировать свои типы свойств через процесс PKCS.

♦ **CK_KEY_TYPE**

CK_KEY_TYPE – это значение, которое указывает на тип ключа. Описывается следующим образом:

```
typedef CK_ULONG CK_KEY_TYPE;
```

Типы ключей описываются вместе с объектами и механизмами, которые используют их. Тип ключа указывается в объекте через атрибут объекта CKA_KEY_TYPE.

Могут также указываться значения для данного типа, определяемые поставщиком.

```
CKK_VENDOR_DEFINED
```

Типы ключей **CKK_VENDOR_DEFINED**, а также те, что указаны выше, постоянно зарезервированы для поставщиков носителя. В целях совместимости поставщики должны регистрировать свои типы ключей через процесс PKCS.

♦ **CK_CERTIFICATE_TYPE**

CK_KEY_TYPE – значение, которое указывает на тип сертификата. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_CERTIFICATE_TYPE;
```

Типы сертификатов описываются вместе с объектами и механизмами, которые используют их. Тип сертификата указывается в объекте через атрибут объекта CKA_CERTIFICATE_TYPE.

Могут также указываться значения для данного типа, определяемые поставщиком.

```
CKC_VENDOR_DEFINED
```

Типы сертификатов **CKK_VENDOR_DEFINED**, а также те, что указаны выше, постоянно зарезервированы для поставщиков носителя. В целях совместимости поставщики должны регистрировать свои типы сертификатов через процесс PKCS.

♦ **CK_ATTRIBUTE_TYPE**

CK_ATTRIBUTE_TYPE – это значение, которое указывает на тип атрибута. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_ATTRIBUTE_TYPE;
```

Типы атрибутов описываются вместе с объектами и механизмами, которые используют их. Атрибуты указываются в объекте как список элементов значений типов и длины. Обычно они представлены в виде шаблона атрибута.

Могут также указываться значения для данного типа, определяемые поставщиком.

```
CKA_VENDOR_DEFINED
```

Типы атрибутов **CKA_VENDOR_DEFINED**, а также те, что указаны выше, постоянно зарезервированы для поставщиков носителя. В целях совместимости поставщики должны регистрировать свои типы атрибутов через процесс PKCS.

♦ **CK_ATTRIBUTE; CK_ATTRIBUTE_PTR**

CK_ATTRIBUTE – это структура, которая включает тип и длину атрибута. Данная структура задается следующим образом:

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

Поля данной структуры имеют следующие значения:

type	тип атрибута;
pValue	указатель на значение атрибута;
ulValueLen	длина значения в байтах.

Если у атрибута нет значения, то *ulValueLen* = 0, а значение *pValue* не существенно. Совокупность структур **CK_ATTRIBUTE** называется «шаблоном» и используется для создания объектов, управления ими и проведения их поиска. Порядок атрибутов в шаблоне *абсолютно не имеет значения*, даже если шаблон содержит атрибуты поставщика. Следует иметь ввиду, что *pValue* – это «пустой» указатель, облегчающий передачу произвольных значений. Как приложение, так и библиотека Cryptoki должны убедиться в том, что указатель может безопасно направляться на

ожидаемый тип (т. е. без ошибок выравнивания по границе слов).

CK_ATTRIBUTE_PTR – указатель на **CK_ATTRIBUTE**.

♦ **CK_DATE**

CK_DATE – это структура, которая определяет дату. Данная структура задается следующим образом:

```
typedef struct CK_DATE {
  CK_CHAR year[4];
  CK_CHAR month[2];
  CK_CHAR day[2];
} CK_DATE;
```

Поля данной структуры имеют следующие значения:

year год («1900» – «9999»);
month месяц («01» – «12»);
day день («01» – «31»).

Поля содержат числовые символы из набора символов таблицы 3, а не байтовые буквенные значения.

Если объект Cryptoki содержит атрибут такого типа, а значение по умолчанию этого атрибута указывается как «пустое», то библиотеки Cryptoki выставят значение атрибута *ulValueLen* на 0.

Внимание! Реализации предыдущих версий Cryptoki могли использовать и другие методы для определения «пустого» атрибута типа CK_DATE, а следовательно, приложению, которому нужно взаимодействовать с данными библиотеками, необходимо быть гибким в отношении того, что они могут принимать в качестве пустого значения.

9.5 Типы данных для механизмов

Cryptoki поддерживает следующие типы данных для описания механизмов, а также их параметры:

♦ **CK_MECHANISM_TYPE; CK_MECHANISM_TYPE_PTR**

CK_MECHANISM_TYPE – значение, которое указывает на тип механизма. Данная структура задается следующим образом:

```
typedef CK_ULONG CK_MECHANISM_TYPE;
```

Типы механизмов задаются вместе с объектами и описаниями механизмов, которые используют их. Могут также задаваться значения для данного типа, определяемые поставщиком.

```
CKM_VENDOR_DEFINED
```

Типы механизмов **CKM_VENDOR_DEFINED**, а также те, что указаны выше, постоянно зарезервированы для поставщиков носителя. В целях совместимости поставщики должны регистрировать свои типы механизмов через процесс PKCS.

CK_MECHANISM_TYPE_PTR – указатель на **CK_MECHANISM_TYPE**.

♦ **CK_MECHANISM; CK_MECHANISM_PTR**

CK_MECHANISM – это структура, которая описывает определенный механизм, а также любые параметры, которые требуются для него. Данная структура задается следующим образом:

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

Поля данной структуры имеют следующие значения:

Mechanism тип механизма;
pParameter указатель на параметр, если он требуется механизмом;
ulParameterLen длина параметра в байтах.

pParameter – это «пустой» указатель, облегчающий передачу произвольных значений. Как приложение, так и библиотека Cryptoki должны убедиться в том, что указатель может безопасно направляться на ожидаемый тип (т. е. без ошибок выравнивания по границе слов).

CK_MECHANISM_PTR – это указатель на **CK_MECHANISM**.

♦ **CK_MECHANISM_INFO; CK_MECHANISM_INFO_PTR**

CK_MECHANISM_INFO – это структура, которая предоставляет информацию о конкретном механизме. Данная структура задается следующим образом:

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

Поля данной структуры имеют следующие значения:

ulMinKeySize минимальный размер ключа для механизма (измеряется он в битах или байтах – зависит от механизма);

ulMaxKeySize максимальный размер ключа для механизма (измеряется он в битах или байтах – зависит от механизма);

flags флаги, указывающие возможности механизма.

Для некоторых механизмов величины в полях *ulMinKeySize* и *ulMaxKeySize* не имеют значения.

В таблице 13 описываются поля *флагов*.

Таблица 13 – Информационные флаги механизма

Флаг	Маска	Значение
CKF_HW	0x00000001	Истинно, если механизм выполняется устройством; ложно, если механизм выполняется на программном уровне
CKF_ENCRYPT	0x00000100	Истинно, если механизм может использоваться с функцией **C_EncryptInit**
CKF_DECRYPT	0x00000200	Истинно, если механизм может использоваться с функцией **C_DecryptInit**
CKF_DIGEST	0x00000400	Истинно, если механизм может использоваться с функцией **C_DigestInit**
CKF_SIGN	0x00000800	Истинно, если механизм может использоваться с функцией **C_SignInit**
CKF_SIGN_RECOVER	0x00001000	Истинно, если механизм может использоваться с функцией **C_SignRecoverInit**
CKF_VERIFY	0x00002000	Истинно, если механизм может использоваться с функцией **C_VerifyInit**
CKF_VERIFY_RECOVER	0x00004000	Истинно, если механизм может использоваться с функцией **C_VerifyRecoverInit**
CKF_GENERATE	0x00008000	Истинно, если механизм может использоваться с функцией **C_GenerateKey**
CKF_GENERATE_KEY_PAIR	0x00010000	Истинно, если механизм может использоваться с функцией **C_GenerateKeyPair**
CKF_WRAP	0x00020000	Истинно, если механизм может использоваться с функцией **C_WrapKey**
CKF_UNWRAP	0x00040000	Истинно, если механизм может использоваться с функцией **C_UnwrapKey**
CKF_DERIVE	0x00080000	Истинно, если механизм может использоваться с функцией **C_DeriveKey**
CKF_EXTENSION	0x80000000	Истинно, если у флагов есть расширение, ложно, если расширения нет. Для данной версии это поле должно содержать значение «ложно».

CK_MECHANISM_INFO_PTR – указатель на **CK_MECHANISM_INFO**.

9.6 Типы функций

Cryptoki представляет информацию о функциях с использованием следующих типов данных:

♦ **CK_RV**

CK_RV – это величина, указывающая на значение, возвращаемое функцией Cryptoki. Данная величина задается следующим образом:

```
typedef CK_ULONG CK_RV;
```

Могут быть также заданы значения для данного типа данных, определяемые поставщиком.

```
CKR_VENDOR_DEFINED
```

В 11.1 определены величины каждого значения **CK_RV.** Возвращаемые значения **CKR_VENDOR_ DEFINED**, а также те, что указаны выше, постоянно зарезервированы для поставщиков носителя.
В целях совместимости поставщики должны регистрировать свои возвращаемые значения через процесс PKCS.

♦ **CK_NOTIFY**

CK_NOTIFY – тип указателя на функцию, который Cryptoki использует для отзыва уведомления. Данная величина задается следующим образом:

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

Аргументы функции отзыва уведомления имеют следующие значения:

hSession	идентификатор сеанса, выполняющего отзыв;
event	тип отзыва уведомления;
pApplication	значение, задаваемое приложением. То же значение, что было передано **C_OpenSession** для открытия сеанса, выполняющего отзыв.

♦ **CK_C_XXX**

Cryptoki также задает все семейство других типов указателей функции. Для каждой функции **C_XXX** в Cryptoki API (для получения подробной информации о каждой из них см. раздел 10.12) Cryptoki задает тип **CK_C_XXX**, который является указателем на функцию с такими же аргументами и возвращаемыми значениями, как у **C_XXX**. Правильно выставленная переменная типа **CK_C_XXX** может быть использована приложением для вызова функции Cryptoki **C_XXX.**

♦ **CK_FUNCTION_LIST; CK_FUNCTION_LIST_PTR; CK_FUNCTION_LIST_PTR_PTR**

CK_FUNCTION_LIST – структура, которая содержит информацию о версии Cryptoki, а также указатель на каждую функцию в Cryptoki API. Данная структура задается следующим образом:

```
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;
```

Каждая библиотека Cryptoki имеет статическую структуру **CK_FUNCTION_LIST**, а указатель на нее (или на ее копию, которая также принадлежит библиотеке) может быть получен при помощи функции **C_GetFunctionList** (см. 11.2). Величина, на которую указывает данный указатель, может использоваться приложением, чтобы быстро выяснить, где в Cryptoki API находится исполняемый код для каждой функции. *Каждая функция в Cryptoki API должна иметь точку входа, которая определена в структуре **CK_FUNCTION_LIST** библиотеки Cryptoki.* Если определенная функция в Cryptoki API не поддерживается библиотекой, то указатель данной функции в библиотеке структуры

CK_FUNCTION_LIST должен указывать на ответвление функции, которое просто выдает значение CKR_FUNCTION_NOT_SUPPORTED.

Приложение может иметь, а может не иметь возможности изменения статической структуры **CK_FUNCTION_LIST** библиотеки Cryptoki. Независимо от того, располагает ли оно такой возможностью, приложение ни в коем случае не должно пытаться делать это.

CK_FUNCTION_LIST_PTR – это указатель на **CK_FUNCTION_LIST**.
CK_FUNCTION_LIST_PTR_PTR – это указатель на **CK_FUNCTION_LIST_PTR**.

9.7 Типы, связанные с блокировкой

Типы данных, описание которых приводится в настоящем разделе, предусмотрены исключительно для приложений, которым нужен доступ в Cryptoki одновременно из нескольких потоков. *Приложениям, которые не будут осуществлять таких операций, не нужно использовать ни один из этих типов.*

♦ **CK_CREATEMUTEX**

CK_CREATEMUTEX – тип указателя на функцию приложения, которая создает новый флаг и возвращает на него указатель. Данная структура задается следующим образом:

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

Вызов функции **CK_CREATEMUTEX** возвращает указатель на новый флаг в месте, указанном ему в поле *ppMutex*. Такая функция должна выдавать одно из следующих значений: CKR_OK, CKR_GENERAL_ ERROR, CKR_HOST_MEMORY.

♦ **CK_DESTROYMUTEX**

CK_DESTROYMUTEX – это тип указателя на функцию приложения, которая уничтожает существующий флаг. Данная структура задается следующим образом:

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

Аргумент функции **CK_DESTROYMUTEX** – указатель на флаг, который необходимо уничтожить. Такая функция должна выдавать одно из следующих значений: CKR_OK, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MUTEX_BAD.

♦ **CK_LOCKMUTEX и CK_UNLOCKMUTEX**

CK_LOCKMUTEX – тип указателя на функцию приложения, которая блокирует существующий флаг. **CK_UNLOCKMUTEX** – это тип указателя на функцию приложения, которая разблокирует существующий флаг. Правильным считается следующее поведение таких функций:

- если функция **CK_LOCKMUTEX** вызывается в отношении флага, который не заблокирован, то поток вызова получает блокировку на этом флаге и возвращается;
- если функция **CK_LOCKMUTEX** вызывается в отношении флага, который заблокирован любым другим потоком, вызывающий поток блокируется и ожидает, пока флаг будет разблокирован;
- если функция **CK_LOCKMUTEX** вызывается в отношении флага, который заблокирован потоком, отправляющим вызов, поведение вызова функции не определено;
- если функция **CK_UNLOCKMUTEX** вызывается в отношении флага, который заблокирован потоком, отправляющим вызов, этот флаг будет разблокирован, а вызов возратится. Кроме того:
- если только один поток блокировался именно на этом флаге, тогда этот поток перестает блокироваться, получает сигнал блокировки флага и его вызов **CK_LOCKMUTEX** возвращается;
- если несколько потоков блокировались именно на этом флаге, тогда некоторым образом отбирается один из блокирующихся потоков. Этот избранный поток прекращает блокироваться, получает сигнал блокировки флага, и его вызов **CK_LOCKMUTEX** возвращается. Все остальные потоки, которые блокировались на данном флаге, продолжают оставаться заблокированными;
- если функция **CK_UNLOCKMUTEX** вызывается в отношении флага, который не заблокирован, тогда вызов функции возвращает сообщение с кодом ошибки CKR_MUTEX_NOT_LOCKED;
- если функция **CK_UNLOCKMUTEX** вызывается в отношении флага, который заблокирован некоторым другим потоком, не отправляющим вызов, поведение вызова функции не определено.

CK_LOCKMUTEX задается следующим образом:

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

Аргумент функции **CK_DESTROYMUTEX** – указатель на объект флага, который необходимо заблокировать. Такая функция должна выдавать одно из следующих значений: CKR_OK, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MUTEX_BAD.

CK_UNLOCKMUTEX задается следующим образом:

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

Аргумент функции **CK_DESTROYMUTEX** – указатель на флаг, который необходимо разблокировать. Такая функция должна выдавать одно из следующих значений: CKR_OK, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MUTEX_BAD, CKR_MUTEX_NOT_LOCKED.

♦ **CK_C_INITIALIZE_ARGS; CK_C_INITIALIZE_ARGS_PTR**

CK_C_INITIALIZE_ARGS – структура, которая содержит дополнительные аргументы функции **C_Initialize.** В данной версии Cryptoki все эти дополнительные аргументы связаны с тем, как библиотека работает с потоками. **CK_C_INITIALIZE_ARGS** задается следующим образом:

```
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 данной структуры имеют следующие значения:

CreateMutex указатель на функцию, которая будет использоваться для создания объектов флага;

DestroyMutex указатель на функцию, которая будет использоваться для уничтожения объектов флага;

LockMutex указатель на функцию, которая будет использоваться для блокировки объектов флага;

UnlockMutex указатель на функцию, которая будет использоваться для разблокировки объектов флага;

flags бит-флаги, задающие опции функции **C_Initialize**; описание данных флагов приводится ниже;

pReserved зарезервировано для использования в будущем. В данной версии Cryptoki данное поле должно содержать NULL_PTR.

В таблице 14 описаны поля *флагов*.

Таблица 14 – Флаги параметров функции C_Initialize

Флаг	Маска	Значение
CKF_LIBRARY_CANT_CREATE_OS_THREADS	0x00000001	Истинно, если потоки приложения, которые передают вызовы к библиотеке, *не* могут использовать собственные вызовы операционной системы для порождения новых потоков; ложно, если могут
CKF_OS_LOCKING_OK	0x00000002	Истинно, если библиотека может для блокирования использовать собственную потоковую модель операционной системы; ложно в обратном случае

CK_C_INITIALIZE_ARGS_PTR – указатель на **CK_C_INITIALIZE_ARGS**.

10 Объекты

Cryptoki различает несколько классов объектов, как определено в типе данных **CK_OBJECT_CLASS.** Объект состоит из набора атрибутов, каждый из которых имеет заданное значение. У каждого из принадлежащих объекту атрибутов есть только одно значение. На рисунке 5 наглядно отображена иерархия высокого уровня объектов Cryptoki, а также некоторые атрибуты, которые поддерживаются данными объектами.

Рисунок 5 – Иерархия атрибутов объекта

Cryptoki предоставляет функции для создания, уничтожения и копирования объектов целиком, а также функции получения и изменения значений их атрибутов. Некоторые криптографические функции (*например,* **C_GenerateKey**) также создают ключевые объекты для хранения своих результатов.

Объекты в Cryptoki всегда «правильно согласованы» – это означает, что объект всегда содержит все требуемые атрибуты, а все объекты совместимы с момента создания объекта. Так данная система отличается от некоторых объектно-ориентированных систем, в которых у объекта нет атрибутов, кроме, возможно, класса на момент создания этого объекта, и на протяжении некоторого времени он остается неинициализированным. В Cryptoki объекты всегда инициализированы.

В большинстве таблиц раздела 10 приводятся описания каждого атрибута Cryptoki в терминах типов данных значения атрибута, а также значения этого атрибута, среди которых может быть начальное значение по умолчанию. Некоторые типы данных определяются напрямую в Cryptoki (*например,* **CK_OBJECT_CLASS**). Атрибут также может принимать значения следующих типов:

Массив байтов	произвольная строка (массив), состоящая из **CK_BYTE**;
Большое целое	строка, состоящая из **CK_BYTE**, которая представляет целое число без знака произвольного размера, старшим байтом является первый (*например,* целое число 32768 представлено как двухбайтовая строка 0x80 0x00);
Локальная строка	незаполненная строка, состоящая из **CK_CHAR** (см. таблицу 3), не завершающаяся нулем;
Строка RFC2279	незаполненная строка, состоящая из **CK_UTF8CHAR**, не завершающаяся нулем;
Носитель	может содержать несколько одинаковых объектов, т. е. двум или более объектам разрешается иметь абсолютно одинаковые значения всех их атрибутов.

В большинстве случаев у каждого типа объектов в спецификации Cryptoki есть полный и строго очерченный набор атрибутов Cryptoki. Некоторые из этих атрибутов имеют значения по умолчанию, и значения их соответствующих атрибутов не нужно указывать при создании объекта; некоторые из этих значений по умолчанию могут даже быть пустыми строками («»). Тем не менее объект содержит эти атрибуты.

Конкретный объект имеет одно значение для каждого своего атрибута, даже если атрибут является атрибутом, определяемым поставщиком, чье значение лежит за рамками компетенции Cryptoki.

Кроме атрибутов Cryptoki объекты могут иметь дополнительные, определенные поставщиком атрибуты, чьи значения и величины не задаются в Cryptoki.

10.1 Создание, изменение и копирование объектов

Все функции Cryptoki, предназначенные для создания, изменения и копирования объектов, в качестве одного из своих аргументов принимают шаблон, а шаблон задает значения атрибутов. Криптографические функции, которые создают объекты (см. 11.14), могут также сами указывать некоторые дополнительные значения атрибутов; какие из атрибутов имеют значения, дополняемые вызовом криптографической функции, зависит от того, какой криптографический механизм выполняется (см. раздел 12). В любом случае, все необходимые атрибуты, которые поддерживаются классом объекта и которые не имеют значений по умолчанию, должны задаваться при создании объекта либо в шаблоне, либо самой функцией.

10.1.1 Создание объектов

Объекты могут создаваться функциями Cryptoki **C_CreateObject** (см. 11.7), **C_GenerateKey**, **C_GenerateKeyPair**, **C_UnwrapKey** и **C_DeriveKey** (см. 11.14). Кроме того, при копировании существующего объекта (с использованием функции **C_CopyObject**) также создается новый объект, однако создание объекта такого типа рассматривается нами отдельно в 10.1.3.

При попытке создания объекта с использованием одной из следующих функций необходимо предоставить соответствующий шаблон.

1. Если в предоставленном шаблоне указывается значение недействительного атрибута, то попытка будет неудачной с кодом ошибки CKR_ATTRIBUTE_TYPE_INVALID. Атрибут будет действительным, если он является либо одним из атрибутов, описанным в спецификации Cryptoki, либо он является дополнительным атрибутом поставщика, который поддерживается библиотекой и носителем.

2. Если в предоставленном шаблоне указывается недействительное значение действительного атрибута, то попытка будет неудачной с кодом ошибки CKR_ATTRIBUTE_VALUE_INVALID. Описание действительных значений атрибутов Cryptoki приводится в спецификации Cryptoki.

3. Если в предоставленном шаблоне указывается значение атрибута только для чтения, то попытка будет неудачной с кодом ошибки CKR_ATTRIBUTE_READ_ONLY. Указание на то, является ли конкретный атрибут Cryptoki атрибутом только для чтения, напрямую делается в спецификации Cryptoki, однако отдельная библиотека или носитель могут еще более ограничивать использование атрибутов по сравнению с тем, что указано в Cryptoki. Иными словами, атрибут, который Cryptoki не рассматривает как атрибут только для чтения, может быть атрибутом только для чтения при определенных обстоятельствах (т. е. в связи с использованием в комбинации с некоторыми другими атрибутами) для определенной библиотеки или носителя. Очевидно, что вопрос о том, является ли определенный атрибут, который не относится к Cryptoki, атрибутом только для чтения, не входит в область действия Cryptoki.

4. Если значений атрибутов, которые предоставлены в шаблоне вместе с любыми значениями по умолчанию и любыми значениями атрибутов, которые включает в объект сама функция создания объекта, недостаточно для того, чтобы полностью задать создаваемый объект, тогда попытка будет неудачной с кодом ошибки CKR_TEMPLATE_INCOMPLETE.

5. Если значения атрибутов, которые предоставлены в шаблоне вместе с любыми значениями по умолчанию и любыми значениями атрибутов, которые включает в объект сама функция создания объекта, не являются совместимыми, тогда попытка будет неудачной с кодом ошибки CKR_TEMPLATE_INCONSISTENT. Набор значений атрибутов не будет совместимым, если не все его члены могут одновременно обслуживаться *носителем*, хотя при этом каждое значение по отдельности для Cryptoki является действительным. Примером несовместимого шаблона будет использование шаблона, в котором указаны два различных значения для одного и того же атрибута. Еще один пример – это попытка создать объект секретного ключа с использованием атрибута, который подходит для создания разнообразных типов открытых и частных/секретных ключей, но не для секретных ключей. Последний пример – это шаблон с атрибутом, который нарушает требования самого носителя. Последний пример несовместимого шаблона связан с особенностями носителя: на другом носителе такой шаблон мог бы и *не* быть несовместимым.

6. Если в предоставляемом шаблоне конкретный атрибут задается более чем один раз (или если шаблон задает то же самое значение для одного атрибута, который предоставило объекту само приложение), тогда поведение Cryptoki задано не полностью. Попытка создания объекта может или завершиться успешно: будет создан тот же самый объект, который был бы создан, если бы атрибут, заданный несколько раз, был бы на самом деле задан только один раз, или эта попытка может завершиться неудачей с кодом ошибки CKR_TEMPLATE_INCONSISTENT. Разработчикам библиотек предлагается создавать библиотеки таким образом, чтобы они считали, что данный атрибут появляется в данном шаблоне лишь один раз; к разработчикам приложений выдвигается требование никогда не включать определенный атрибут в определенный шаблон более одного раза.

Если к попытке создания объекта можно применить более чем одну из вышеперечисленных ситуаций, то система может выдать любой из кодов ошибки, упомянутых в вышеперечисленных примерах.

10.1.2 Изменение объектов

Объекты можно изменять с использованием функции Cryptoki **C_SetAttributeValue** (см. 11.7). Шаблон, предоставляемый функции **C_SetAttributeValue**, может содержать новые значения для атрибутов, которыми объект уже обладает, значения для атрибутов, которых у объекта еще нет, или и те и другие.

Некоторые атрибуты объекта можно изменять после того, как объект уже был создан, а некоторые – нельзя. Кроме того, атрибуты, которые задает Cryptoki как изменяемые, могут на самом деле *не* являться изменяемыми на некоторых носителях. Это означает, что, если атрибут Cryptoki описан как изменяемый, на самом деле имеется в виду, что он является изменяемым только в тех случаях, *когда речь идет о спецификации Cryptoki.* На самом деле, определенный носитель может и не поддерживать изменение некоторых таких атрибутов. Более того, является или не является конкретный атрибут объекта изменяемым на конкретном носителе, может зависеть от значений определенных атрибутов объекта. Например, атрибут **CKA_SENSITIVE** объекта секретного ключа может изменяться с CK_FALSE на CK_TRUE, но не в обратном порядке.

Все сценарии из 10.1.1 и коды ошибок, которые они выдают, применяются к изменению объектов с использованием **C_SetAttributeValue**, за исключением возможной неполноты шаблона.

10.1.3 Копирование объектов

Объекты можно копировать с использованием функции Cryptoki **C_CopyObject** (см. 11.7). В процессе копирования объектов функция **C_CopyObject** также изменяет и атрибуты создаваемых копий в соответствии с шаблоном, предоставленным приложением.

Атрибуты Cryptoki, которые можно изменять в процессе проведения операции **C_CopyObject** – это те же атрибуты Cryptoki, которые описаны как изменяемые, плюс три специальных атрибута **CKA_TOKEN**, **CKA_PRIVATE** и **CKA_MODIFIABLE**. Именно эти атрибуты являются изменяемыми в процессе проведения операции **C_CopyObject** только в тех случаях, *когда идет речь о спецификации Cryptoki.* Конкретный носитель может на самом деле и не поддерживать изменение некоторых таких атрибутов в процессе проведения операции **C_CopyObject**. Более того, является или не является конкретный атрибут объекта изменяемым в процессе проведения операции **C_CopyObject** на конкретном носителе, может зависеть от значений определенных атрибутов объекта. Например, атрибут **CKA_SENSITIVE** объекта секретного ключа может изменяться с CK_FALSE на CK_TRUE в процессе проведения операции **C_CopyObject**, но не в обратном порядке.

Все сценарии из 10.1.1 и коды ошибок, которые они выдают, применяются к изменению объектов в процессе проведения операции **C_CopyObject**, за исключением возможной неполноты шаблона.

10.2 Общие атрибуты

Таблица 15 – Общие сноски в таблицах атрибутов объектов

[1] Указывается, когда объект создается с использованием **C_CreateObject**.
[2] *Не* указывается, когда объект создается с использованием **C_CreateObject**.
[3] Указывается, когда объект генерируется с использованием **C_GenerateKey** или **C_GenerateKeyPair**.
[4] *Не* указывается, когда объект генерируется с использованием **C_GenerateKey** или **C_GenerateKeyPair**.
[5] Должно указываться, когда объект распаковывается с использованием **C_UnwrapKey**.
[6] *Не* указывается, когда объект создается с использованием **C_CreateObject**.
[7] Не может открываться, если атрибут объекта **CKA_SENSITIVE** выставлен на CK_TRUE или его атрибут **CKA_EXTRACTABLE** выставлен на CK_FALSE.
[8] Может изменяться, если объект создан с использованием вызова функции **C_SetAttributeValue** или в процессе копирования объекта с использованием вызова функции **C_CopyObject**. Однако в некоторых случаях определенный носитель может и не разрешить изменение атрибута в процессе выполнения вызова функции **C_CopyObject**.
[9] Значение по умолчанию является специфическим для конкретного носителя и может зависеть от значений других атрибутов.
[10] Может выставляться на CK_TRUE только пользователем с правами SO.
[11] Attribute cannot be changed once set to CK_TRUE. Становится атрибутом только для чтения.
[12] Атрибут не изменяется после того как выставлен на CK_FALSE. Становится атрибутом только для чтения.

Таблица 16 – Общие атрибуты объектов

Атрибут	Тип данных	Значение
CKA_CLASS [1]	CK_OBJECT_CLASS	Класс (тип) объекта
См. сноски в таблице 15.		

В таблице 16 указаны атрибуты, которые являются общими для всех объектов.

10.3 Объекты свойств оборудования

10.3.1 Определения

В настоящем разделе определяется класс объекта CKO_HW_FEATURE для типа CK_OBJECT_CLASS, как объект, используемый в атрибуте объектов CKA_CLASS.

10.3.2 Обзор

Объекты свойств оборудования (**CKO_HW_FEATURE**) указывают на свойства устройств. Они предоставляют методику легкого расширения для введения новых, основанных на значениях свойств в интерфейс Cryptoki.

При поиске объектов с использованием **C_FindObjectsInit** и **C_FindObjects** объекты свойств оборудования не будут возвращаться, если атрибут **CKA_CLASS** в шаблоне не имеет значение **CKO_HW_FEATURE**. Это предотвращает возможность нахождения приложениями, написанными для предыдущих версий Cryptoki, непонятных для них объектов.

Таблица 17 – Общие атрибуты свойств оборудования

Атрибут	Тип данных	Значение
CKA_HW_FEATURE_TYPE [1]	CK_HW_FEATURE	Свойство (тип) оборудования
См. сноски в таблице 15.		

В таблице 17 указаны атрибуты, которые являются общими для всех объектов.

10.3.3 Часы

10.3.3.1 Определение

Атрибут **CKA_HW_FEATURE_TYPE** принимает значение **CKH_CLOCK** типа **CK_HW_FEATURE.**

10.3.3.2 Описание

Объекты часов представляют собой часы в реальном масштабе времени, которые встроены в устройство. Это те же часы, что и в поле **utcTime** структуры **CK_TOKEN_INFO.**

Таблица 18 – Атрибуты объектов часов

Атрибут	Тип данных	Значение
CKA_VALUE	CK_CHAR[16]	Текущее время в виде символьной строки длиной в 16 символов в формате YYYYMMDDhhmmssxx (4 символа для года; по два символа на месяц, день, час, минуты и секунды; 2 дополнительных зарезервированных пустых символа ‘0’)

Атрибут **CKA_VALUE** может быть выставлен с использованием функции **C_SetAttributeValue,** если это разрешено устройством. Сеанс, который используется для выставления времени, должен быть зарегистрирован в устройстве. Устройство может потребовать, чтобы SO прошел регистрацию для изменения времени. _**SetAttributeValue** приведет к выдаче сообщения с кодом ошибки CKR_USER_ NOT_LOGGED_IN, которое будет указывать на то, что для выставления данного значения требуется другой тип пользователя.

10.3.4 Объекты монотонного счетчика

10.3.4.1 Определение

Атрибут **CKA_HW_FEATURE_TYPE** принимает значение **CKH_MONOTONIC_COUNTER** типа **CK_HW_FEATURE.**

10.3.4.2 Описание

Объекты монотонных счетчиков представляют собой аппаратные счетчики, которые имеются на устройствах. Гарантируется, что значение счетчика увеличивается каждый раз, когда считывается его величина, но не обязательно увеличивается на один. Это может использоваться приложениями для генерации серийных номеров в целях подтверждения уникальности каждого носителя.

Таблица 19 – Атрибуты монотонных счетчиков

Атрибут	Тип данных	Значение
CKA_RESET_ON_INIT [1]	CK_BBOOL	Счетчик будет переустановлен на предыдущее возвращенное значение, если носитель инициализируется с использованием **C_InitializeToken**
CKA_HAS_RESET [1]	CK_BBOOL	Значение счетчика переустанавливалось по меньшей мере один раз в какой-то момент времени
CKA_VALUE [1]	Byte Array	Текущая версия монотонного счетчика. Значение выдается в обратном порядке байтов (старший байт передается сначала)
[1] Только для чтения.		

Атрибут **CKA_VALUE** не может выставляться клиентом.

10.3.5 Объекты пользовательского интерфейса

10.3.5.1 Определение

Атрибут **CKA_HW_FEATURE_TYPE** принимает значение **CKH_USER_INTERFACE** типа **CK_HW_FEATURE.**

10.3.5.2 Описание

Объекты пользовательского интерфейса указывают на возможности устройства по представлению информации.

Таблица 20 – Атрибуты объектов пользовательского интерфейса

Атрибут	Тип данных	Значение
CKA_PIXEL_X	CK_ULONG	Разрешение экрана (в пикселях) по оси абсцисс (например, 1280)
CKA_PIXEL_Y	CK_ULONG	Разрешение экрана (в пикселях) по оси ординат (например, 1024)
CKA_RESOLUTION	CK_ULONG	Число точек на дюйм
CKA_CHAR_ROWS	CK_ULONG	Для дисплеев, основанных на символах, количество символьных рядов (например, 24)
CKA_CHAR_COLUMNS	CK_ULONG	Для дисплеев, основанных на символах, количество столбцов символов (например, 80). Если в дисплее используется пропорциональный шрифт, этот атрибут указывает на ширину дисплея в буквах «m», см. CC/PP Struct
CKA_COLOR	CK_BBOOL	Поддержка цветов
CKA_BITS_PER_PIXEL	CK_ULONG	Количество битов в информации о цвете или количестве оттенков серого на пиксель
CKA_CHAR_SETS	Строка RFC 2279	Строка с указанием на поддерживаемые наборы символов, согласно определению наборов IANA MIBenum (www.iana.org). Поддерживаемые наборы символов записываются через «;». Например, носитель, поддерживающий iso-8859-1 и us-ascii, выставил бы значение данного атрибута на «4;3»

Окончание таблицы 20

Атрибут	Тип данных	Значение
CKA_ENCODING_METHODS	Строка RFC 2279	Строка с указанием на поддерживаемые методы кодирования передаваемой информации, согласно определению IANA (www.iana.org). Поддерживаемые методы записаны через «;». Например, носитель, поддерживающий 7-битные, 8-битные методы и base64, мог бы выставить значение данного атрибута на «7bit;8bit;base64»
CKA_MIME_TYPES	Строка RFC 2279	Строка с указанием на поддерживаемые (представляемые) MIME-типы, согласно определению IANA (www.iana.org). Поддерживаемые типы записаны через «;». Например, носитель, поддерживающий MIME-типы «a/b», «a/c» and «a/d», выставил бы значение этого атрибута на «a/b;a/c;a/d»

Данная подборка атрибутов и связанных с ними типов данных была сделана в связи с попыткой как можно больше ориентироваться на RFC 2534 и CC/PP Struct.

Когда информация отсутствует или не применима, для атрибутов, основанных на CK_ULONG, может использоваться специальное значение CK_UNAVAILABLE_INFORMATION.

Ни одно из значений атрибутов не может выставляться приложением.

Значение атрибута **CKA_ENCODING_METHODS** может использоваться в случаях, когда приложению нужно отправить на носитель объекты MIME с закодированным содержанием.

10.4 Объекты хранения

Это не класс объектов, а следовательно, не требуется определение CKO_. Это категория классов объектов с общими атрибутами для следующих классов объектов.

Таблица 21 – Общие атрибуты объектов хранения

Атрибут	Тип данных	Значение
CKA_TOKEN	CK_BBOOL	CK_TRUE, истинно, если объект является объектом носителя; CK_FALSE, ложно, если объект является объектом сеанса. По умолчанию выставляется CK_FALSE
CKA_PRIVATE	CK_BBOOL	CK_TRUE, истинно, если объект является секретным объектом; CK_FALSE, ложно, если объект является открытым объектом. Значение по умолчанию является специфическим для конкретного носителя и может зависеть от значений других атрибутов объекта
CKA_MODIFIABLE	CK_BBOOL	CK_TRUE, истинно, если объект можно изменить. По умолчанию выставляется CK_TRUE
CKA_LABEL	Строка RFC2279	Описание объекта (по умолчанию этот атрибут пустой)

Только атрибут **CKA_LABEL** можно изменять после создания объекта (атрибуты **CKA_TOKEN**, **CKA_PRIVATE** и **CKA_MODIFIABLE** можно изменять в процессе копирования объекта).

Атрибут **CKA_TOKEN** указывает на то, является ли объект объектом носителя или сеанса.

Когда атрибут **CKA_PRIVATE** выставлен на CK_TRUE, пользователь не сможет получить доступ к объекту до тех пор, пока не пройдет аутентификацию в объекте.

Значение атрибута **CKA_MODIFIABLE** определяет, является ли объект объектом только для чтения или нет. Могут быть варианты, при которых неизменяемый объект можно удалить, однако есть варианты, когда его удалить нельзя.

Атрибут **CKA_LABEL** предназначен для оказания помощи пользователям при просмотре и редактировании.

10.5 Объекты данных

10.5.1 Определения

В настоящем разделе определяется класс объекта CKO_DATA для типа CK_OBJECT_CLASS, как используемого в атрибуте объектов CKA_CLASS.

10.5.2 Обзор

Объекты данных (класс объекта **CKO_DATA**) хранят информацию, заданную приложением. Кроме предоставления доступа к ним, Cryptoki не придает объектам данных никакого дополнительного значения. В таблице 22 приведен список атрибутов, которые поддерживаются объектами данных, кроме общих атрибутов, которые определены для данного класса.

Таблица 22 – Атрибуты объектов данных

Атрибут	Тип данных	Значение
CKA_APPLICATION	Строка RFC2279	Описание приложения, которое управляет объектом (по умолчанию этот атрибут пустой)
CKA_OBJECT_ID	Совокупность байтов	DER-шифрование идентификатора объекта, который указывает на тип объекта данных (по умолчанию этот атрибут пустой)
CKA_VALUE	Совокупность байтов	Значение объекта (по умолчанию этот атрибут пустой)

Атрибут **CKA_APPLICATION** обеспечивает для приложений средство определения принадлежности объектов данных, которыми они управляют. Однако Cryptoki не предоставляет средств, которые удостоверяли бы, что только конкретное приложение имеет доступ к отдельному объекту данных.

Атрибут **CKA_OBJECT_ID** предоставляет приложению независимый и расширяемый способ определения типа значения объекта данных. Cryptoki не предоставляет средств, которые удостоверяли бы, что идентификатор объекта данных соответствует значению данных.

Ниже приводится пример шаблона, содержащего атрибуты для создания объекта данных:

```
CK_OBJECT_CLASS class = CKO_DATA;
CK_UTF8CHAR label[] = “Объект данных”;
CK_UTF8CHAR application[] = “Приложение”;
CK_BYTE data[] = “Пример данных”;
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)}
};
```

10.6 Объекты сертификата

10.6.1 Определения

В настоящем разделе класс объекта CKO_CERTIFICATE определяется как класс типа CK_OBJECT_CLASS, используемого в атрибуте объектов CKA_CLASS.

10.6.2 Обзор

Объекты сертификатов (класс объектов **CKO_CERTIFICATE**) хранят открытые ключи или сертификаты атрибутов. Кроме предоставления доступа к объектам сертификатов, Cryptoki не придает сертификатам никакого дополнительного значения. В таблице 23 приведены определения общих атрибутов объектов сертификатов, кроме общих атрибутов, определенных для данного класса объектов.

Таблица 23 – Общие атрибуты объектов сертификатов

Атрибут	Тип данных	Значение
CKA_CERTIFICATE_TYPE [1]	CK_CERTIFICATE_TYPE	Тип сертификата
CKA_TRUSTED [10]	CK_BBOOL	Сертификат может быть доверен приложению, которое создавало его
CKA_CERTIFICATE_CATEGORY	CK_ULONG	Категоризация сертификата: 0 = не указано (значение по умолчанию); 1 = пользователь носителя; 2 = удостоверяющий центр; 3 = иное лицо
CKA_CHECK_VALUE	Совокупность байтов	Контрольная сумма
CKA_START_DATE	CK_DATE	Начальная дата сертификата (по умолчанию этот атрибут пустой)
CKA_END_DATE	CK_DATE	Конечная дата сертификата (по умолчанию этот атрибут пустой)
См. сноски в таблице 15.		

Атрибут **CKA_CERTIFICATE_TYPE** не изменяется после создания объекта. Данная версия Cryptoki поддерживает следующие типы сертификатов:

- сертификат открытого ключа X.509;
- сертификат открытого ключа WTLS;
- атрибут сертификата X.509.

Атрибут **CKA_TRUSTED** не может быть выставлен приложением на CK_TRUE. Он должен быть выставлен приложением, инициализирующим носитель или SO носителя. Доверенные сертификаты не могут быть изменены.

Атрибут **CKA_CERTIFICATE_CATEGORY** используется для указания того, является ли сохраняемый сертификат сертификатом пользователя, для которого соответствующий личный сертификат является доступным на носителе («пользователь носителя»), сертификатом УЦ («удостоверяющий центр») или конечным сертификатом («иного лица»). Данный атрибут не изменяется после создания объекта.

Атрибуты **CKA_CERTIFICATE_CATEGORY** и **CKA_TRUSTED** вместе будут использоваться для сопоставления с категориями сертификатов. В CDF-описании сертификатов сохраняемый сертификат помечен категорией «пользователь носителя». В CDF-описаниях «trustedCertificates» или «usefulCertificates» сохраняемый сертификат будет помечен категорией «удостоверяющий центр» или «иное лицо», в зависимости от того, указывают ли атрибуты CommonCertificateAttribute или **CKA_TRUSTED** на то, принадлежит он к CDF-описаниям trustedCertificates или usefulCertificates.

CKA_CHECK_VALUE: Значение данного сертификата выводится из сертификата по первым трем байтам значения хэш-функции SHA-1 атрибута CKA_VALUE объекта сертификата.

Атрибуты **CKA_START_DATE** и **CKA_END_DATE** используются только для ссылки; Cryptoki не придает им никакого дополнительного значения. Когда они присутствуют, приложение отвечает за то, чтобы выставить их на значения, которые соответствуют полям сертификата «не до» и «не после» (если таковые существуют).

10.6.3 Объект сертификата открытого ключа X.509

Объекты сертификата X.509 (тип сертификата **CKC_X_509**) содержит сертификаты открытых ключей X.509. В таблице 24 приведены определения атрибутов объектов сертификатов X.509, кроме общих атрибутов, определенных для данного класса объектов.

Таблица 24 – Атрибуты объектов сертификатов X.509

Атрибут	Тип данных	Значение
CKA_SUBJECT [1]	Совокупность байтов	DER-шифрование имени субъекта сертификата
CKA_ID	Совокупность байтов	Идентификатор ключа для пары открытый/секретный ключ (по умолчанию этот атрибут пустой)
CKA_ISSUER	Совокупность байтов	DER-шифрование имени структуры, выдавшей сертификат (по умолчанию этот атрибут пустой)
CKA_SERIAL_NUMBER	Совокупность байтов	DER-шифрование серийного номера сертифи-ката лица (по умолчанию этот атрибут пустой)
CKA_VALUE [2]	Совокупность байтов	BER-шифрование сертификата
CKA_URL [3]	Строка RFC2279	Если этот атрибут не пустой, он содержит URL, где можно получить полный сертификат (по умолчанию этот атрибут пустой)
CKA_HASH_OF_SUBJECT_PUBLIC_KEY [4]	Совокупность байтов	Хэш SHA-1 открытого ключа субъекта (по умолчанию этот атрибут пустой)
CKA_HASH_OF_ISSUER_PUBLIC_KEY [4]	Совокупность байтов	Хэш SHA-1 открытого ключа лица, выдавшего сертификат (по умолчанию этот атрибут пустой)
CKA_JAVA_MIDP_SECURITY_DOMAIN	CK_ULONG	Домен безопасности Java MIDP: 0 = не указано (значение по умолчанию); 1 = производитель; 2 = оператор; 3 = третье лицо

[1] Указывается при создании объекта.
[2] Указывается при создании объекта. Не должен быть пустым, если пустым является CKA_URL.
[3] Не должен быть пустым, если пустым является CKA_VALUE.
[4] Может быть пустым, только если пустым является CKA_URL.

Только атрибуты **CKA_ID**, **CKA_ISSUER** и **CKA_SERIAL_NUMBER** можно изменять после создания объекта.

Атрибут **CKA_ID** предназначен как средство распознавания множества пар открытый ключ/секретный ключ, которые хранятся в одном и том же объекте (независимо от того, хранятся ли они в одном носителе или нет. Так как ключи распознаются по имени объекта, а также по идентификатору, возможно, что ключи к различным объектам могут иметь то же значение **CKA_ID**, не приводя к неоднозначности.

В интересах совместимости предусмотрено, что имя объекта и идентификатор ключа сертификата будут совпадать с такими же показателями соответствующих секретных и открытых ключей (хотя и не требуется, чтобы все они хранились в одном и том же носителе). Однако Cryptoki не настаивает на такой привязке или даже на уникальности идентификатора ключа для конкретного субъекта; в частности, приложение может оставить идентификатор ключа пустым.

Атрибуты **CKA_ISSUER** и **CKA_SERIAL_NUMBER** предназначены для совместимости с PKCS #7 и со стандартом Privacy Enhanced Mail (RFC1421). В расширениях версии 3 к сертификатам X.509 идентификатор ключа может находиться в сертификате, значение **CKA_ID** будет являться идентичным идентификатору ключа в таком расширении сертификата, хотя Cryptoki на этом и не настаивает.

Атрибут **CKA_URL** обеспечивает поддержку хранения URL в том месте, где должен храниться сертификат вместо самого сертификата. Хранение URL вместо полного сертификата часто используется в мобильной среде.

Атрибуты **CKA_HASH_OF_SUBJECT_PUBLIC_KEY** и **CKA_HASH_OF_ISSUER_PUBLIC_KEY** используются для хранения хэшей, а также открытых ключей субъекта и органа, выдавшего сертификат. Они особенно важны, когда доступен только URL для сопоставления сертификата с личным ключом, а также при поиске сертификата органа, выдавшего сертификат.

Атрибут **CKA_JAVA_MIDP_SECURITY_DOMAIN** связывает сертификат с доменом безопасности Java MIDP.

Ниже приводится пример шаблона для создания объекта сертификата X.509:

```
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)}
};
```

10.6.4. Объекты сертификата открытого ключа WTLS

Объекты сертификата WTLS (тип сертификата **CKC_WTLS**) содержат сертификаты открытого ключа WTLS. В таблице 25 приведены определения атрибутов объектов сертификатов WTLS в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 25 – Атрибуты объекта сертификата WTLS

Атрибут	Тип данных	Значение
CKA_SUBJECT [1]	Совокупность байтов	WTLS-шифрование (тип идентификатора) субъекта сертификата
CKA_ISSUER	Совокупность байтов	WTLS-шифрование (тип идентификатора) органа, выдавшего сертификат (по умолчанию этот атрибут пустой)
CKA_VALUE [2]	Совокупность байтов	WTLS-шифрование сертификата
CKA_URL [3]	Строка RFC2279	Если этот атрибут не пустой, он содержит URL, где можно получить полный сертификат
CKA_HASH_OF_SUBJECT_PUBLIC_KEY [4]	Совокупность байтов	Хэш SHA-1 открытого ключа субъекта (по умолчанию этот атрибут пустой)
CKA_HASH_OF_ISSUER_PUBLIC_KEY [4]	Совокупность байтов	Хэш SHA-1 открытый ключ органа, выдавшего сертификат (по умолчанию пустой)

[1] Указывается при создании объекта. Может быть пустым, только если пустым является CKA_VALUE.
[2] Указывается при создании объекта. Данный атрибут должен быть пустым, если пустым является атрибут CKA_URL.
[3] Не должен быть пустым, если пустым является CKA_VALUE.
[4] Может быть пустым, только если пустым является CKA_URL.

Только атрибут **CKA_ISSUER** можно изменять после создания объекта.

Описание шифрования атрибутов **CKA_SUBJECT**, **CKA_ISSUER** и **CKA_VALUE** можно найти в [WTLS] (см. раздел 3).

Атрибут **CKA_URL** обеспечивает поддержку хранения URL в том месте, где должен храниться сертификат вместо самого сертификата. Хранение URL вместо полного сертификата часто используется в мобильной среде.

Атрибут **CKA_JAVA_MIDP_SECURITY_DOMAIN** связывает сертификат с доменом безопасности Java MIDP. Атрибуты **CKA_HASH_OF_SUBJECT_PUBLIC_KEY** и **CKA_HASH_OF_ISSUER_PUBLIC_KEY** используются для хранения хэшей, а также открытых ключей субъекта и органа, выдавшего сертификат. Они особенно важны, когда доступен только URL для сопоставления сертификата с личным ключом, а также при поиске сертификата органа, выдавшего сертификат.

Ниже приводится пример шаблона для создания объекта сертификата WTLS:

```
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)}
};
```

10.6.5. Объекты сертификата атрибута X.509

Объекты сертификата атрибута X.509 (тип сертификата **CKC_X_509_ATTR_CERT**) содержат сертификаты атрибута X.509. В таблице 26 приведены определения атрибутов объектов сертификатов атрибутов X.509 в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 26 – Атрибуты объекта сертификата X.509

Атрибут	Тип данных	Значение
CKA_OWNER [1]	Совокупность байтов	DER-шифрование поля субъекта сертификата атрибута. Данный атрибут отличается от атрибута CKA_SUBJECT, который имеется в сертификатах CKC_X_509, поскольку синтаксис и шифрование ASN.1 различны
CKA_AC_ISSUER	Совокупность байтов	DER-шифрование поля субъекта сертификата атрибута органа, выдавшего сертификат. Данный атрибут отличается от атрибута CKA_ISSUER, который имеется в сертификатах CKC_X_509, так как синтаксис и шифрование ASN.1 различны (по умолчанию этот атрибут пустой)
CKA_SERIAL_NUMBER	Совокупность байтов	DER-шифрование серийного номера сертификата (по умолчанию пустой)
CKA_ATTR_TYPES	Совокупность байтов	BER-шифрование последовательности значений идентификаторов объекта, соответствующих типам атрибутов, содержащихся в сертификате. Когда это поле присутствует, оно предоставляет возможность приложениям вести поиск определенного сертификата атрибута без отбора и анализа самого сертификата (по умолчанию пустой)
CKA_VALUE [1]	Совокупность байтов	BER-шифрование сертификата
[1] Атрибут должен задаваться при создании объекта.		

Только атрибуты **CKA_AC_ISSUER**, **CKA_SERIAL_NUMBER** и **CKA_ATTR_TYPES** можно изменять после создания объекта.

Ниже приводится пример шаблона для создания объекта сертификата атрибута X.509:

```
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)}
};
```

10.7 Объекты ключа

10.7.1 Определения

Не существует определения CKO_ для класса объектов базовых ключей, есть определения только для типов ключей, выводимых из базовых.

В настоящем разделе определяются классы объекта CKO_PUBLIC_KEY, CKO_PRIVATE_KEY и CKO_SECRET_KEY для типа CK_OBJECT_CLASS при использовании в атрибуте объектов CKA_CLASS.

10.7.2 Обзор

В объектах ключей хранятся ключи шифрования и аутентификации, которые могут быть открытыми, частными/секретными и секретными ключами. Ниже приводятся общие примечания, которые применяются ко всем таблицам, содержащим описание атрибутов ключей.

Таблица 27 содержит определения атрибутов, которые являются общими для классов открытых, частных/секретных и секретных ключей в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 27 – Общие атрибуты ключа

Атрибут	Тип данных	Значение
CKA_KEY_TYPE [1, 5]	CK_KEY_TYPE	Тип ключа
CKA_ID [8]	Совокупность байтов	Идентификатор ключа (по умолчанию пустой)
CKA_START_DATE [8]	CK_DATE	Дата начала действия ключа (по умолчанию пусто)
CKA_END_DATE [8]	CK_DATE	Дата истечения действия ключа (по умолчанию пусто)
CKA_DERIVE [8]	CK_BBOOL	Выставлен на CK_TRUE, если ключ поддерживает деривацию ключей (т. е. из этого ключа могут выводиться другие ключи (по умолчанию CK_FALSE)
CKA_LOCAL [2, 4, 6]	CK_BBOOL	Выставлен на CK_TRUE, если соблюдается одно из следующих условий: • ключ сгенерирован локально (т. е. на носителе) при помощи вызова функций **C_GenerateKey** или **C_GenerateKeyPair;** • ключ создан при помощи вызова функции **C_CopyObject,** как копия ключа, атрибут которого **CKA_LOCAL** был выставлен на CK_TRUE
CKA_KEY_GEN_MECHANISM [2, 4, 6]	CK_MECHANISM_TYPE	Идентификатор механизма, который используется для генерации материала ключа
CKA_ALLOWED_MECHANISMS	CK_MECHANISM_TYPE _PTR, указатель на массив CK_MECHANISM_TYPE	Список механизмов, которые разрешено использовать с данным ключом. Количество механизмов в массиве – это компонент *ulValueLen* атрибута, разделенный по размеру CK_MECHANISM_TYPE
См. сноски в таблице 15.		

Поле **CKA_ID** предназначено для различения нескольких ключей. В случае с открытыми и секретными ключами данное поле предназначено для управления различными ключами, которые принадлежат одному субъекту; идентификатор для открытого ключа и для соответствующего ему секретного

ключа должны быть одинаковыми. Идентификатор ключа также должен быть одинаковым и для соответствующего сертификата, если таковой существует. Однако Cryptoki не настаивает на таких привязках (дополнительно см. 10.6).

В случае с секретными ключами значение атрибута **CKA_ID** зависит от приложения.

Атрибуты **CKA_START_DATE** и **CKA_END_DATE** используются только для ссылки на них; Cryptoki не придает им никакого особого значения. В частности, он не ограничивает использование ключа в соответствии с датами; этим занимается приложение.

Атрибут **CKA_DERIVE** имеет значение CK_TRUE тогда и только тогда, когда из данного ключа можно вывести другие ключи.

Атрибут **CKA_LOCAL** имеет значение CK_TRUE тогда и только тогда, когда значение ключа было изначально сгенерировано на носителе с использованием вызова функции **C_GenerateKey** или **C_GenerateKeyPair.**

Атрибут **CKA_KEY_GEN_MECHANISM** определяет механизм генерации ключа, который используется для генерации материала ключа. Он содержит действительное значение только в том случае, если атрибут **CKA_LOCAL** имеет значение CK_TRUE. Если **CKA_LOCAL** имеет значение CK_FALSE, то атрибут **CKA_KEY_GEN_MECHANISM** принимает значение CK_UNAVAILABLE_INFORMATION.

10.8 Объекты открытого ключа

Объекты открытых ключей (класс объекта **CKO_PUBLIC_KEY**) содержат открытые ключи. Таблица 28 содержит определения общих атрибутов для всех открытых ключей, кроме общих атрибутов в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 28 – Общие атрибуты открытого ключа

Атрибут	Тип данных	Значение
CKA_SUBJECT [8]	Совокупность байтов	DER-шифрование имени субъекта ключа (по умолчанию пустой)
CKA_ENCRYPT [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает шифрование [9]
CKA_VERIFY [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает верификацию, когда подпись является приложением к данным [9]
CKA_VERIFY_RECOVER [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает верификацию, когда данные восстанавливаются из подписи [9]
CKA_WRAP [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает упаковку (т. е. может использоваться для упаковки других ключей) [9]
CKA_TRUSTED [10]	CK_BBOOL	Ключ может быть доверенным для приложения, которое создавало его. Упаковочный ключ может использоваться для упаковки ключей с использованием CKA_WRAP_WITH_TRUSTED, выставленного на CK_TRUE
CKA_WRAP_TEMPLATE	CK_ATTRIBUTE_PTR	Для упаковочных ключей. Шаблон атрибута должен совпадать с любыми ключами, упакованными с использованием данного упаковочного ключа. Ключи, которые не совпадают, не могут упаковываться. Количество атрибутов в массиве – компонент *ulValueLen* атрибута, разделенный по размеру CK_ATTRIBUTE
См. сноски в таблице 15.		

В интересах совместимости предусмотрено, что имя субъекта и идентификатор ключа для открытого ключа будут совпадать с такими же показателями соответствующего сертификата и секретного ключа. Однако Cryptoki не настаивает на такой привязке, и не требуется, чтобы сертификат и секретный ключ также хранились на носителе.

Для того, чтобы установить соответствие между флагами ISO/IEC 9594-8 (X.509) **keyUsage** для открытых ключей и атрибутами PKCS #11 для открытых ключей, необходимо воспользоваться таблицей 29.

Таблица 29 – Сопоставление флагов использования ключа X.509 с атрибутами Cryptoki для открытых ключей

Флаги использования ключа для открытых ключей в сертификатах открытых ключей X.509	Соответствующие атрибуты Cryptoki для открытых ключей
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

10.9 Объекты частного/секретного ключа

Объекты секретного ключа (класс объекта **CKO_PRIVATE_KEY**) содержат частные/секретные ключи. Таблица 30 содержит определения общих атрибутов для всех частных/секретных ключей, кроме общих атрибутов, определенных для данного класса объектов.

Таблица 30 – Общие атрибуты частного/секретного ключа

Атрибут	Тип данных	Значение
CKA_SUBJECT [8]	Совокупность байтов	DER-шифрование имени субъекта сертификата (по умолчанию пустой)
CKA_SENSITIVE [8, 11]	CK_BBOOL	CK_TRUE, если ключ является уязвимым [9]
CKA_DECRYPT [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает расшифрование [9]
CKA_SIGN [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает электронные подписи, когда подпись является приложением к данным [9]
CKA_SIGN_RECOVER [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает электронные подписи, когда данные можно восстановить из подписи [9]
CKA_UNWRAP [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает распаковку (т. е. может использоваться для распаковки других ключей) [9]
CKA_EXTRACTABLE [8, 12]	CK_BBOOL	CK_TRUE, если ключ является извлекаемым и может упаковываться [9]
CKA_ALWAYS_SENSITIVE [2, 4, 6]	CK_BBOOL	CK_TRUE, если атрибут ключа CKA_SENSITIVE *всегда* был выставлен на CK_TRUE
CKA_NEVER_EXTRACTABLE [2, 4, 6]	CK_BBOOL	CK_TRUE, если атрибут ключа CKA_EXTRACTABLE *никогда не* был выставлен на CK_TRUE
CKA_WRAP_WITH_TRUSTED [11]	CK_BBOOL	CK_TRUE, если ключ может упаковываться только при помощи упаковочного ключа, атрибут CKA_TRUSTED которого выставлен на CK_TRUE. По умолчанию выставляется на CK_FALSE

Окончание таблицы 30

Атрибут	Тип данных	Значение
CKA_UNWRAP_TEMPLATE	CK_ATTRIBUTE_PTR	Для упаковочных ключей. Шаблон атрибута применяется к любым новым ключам, распакованным с использованием данного упаковочного ключа. Любой шаблон, предоставленный пользователем, применяется после этого шаблона, как если бы объект уже был создан. Количество атрибутов в массиве – компонент *ulValueLen* атрибута, разделенный по размеру CK_ATTRIBUTE
CKA_ALWAYS_AUTHENTICATE	CK_BBOOL	Если CK_TRUE, то пользователь должен вводить PIN при любом действии (подпись или расшифрование) с ключом. По умолчанию выставляется на CK_FALSE
См. сноски в таблице 15.		

В интересах совместимости предусмотрено, что имя субъекта и идентификатор ключа для частного/секретного ключа будут совпадать с такими же показателями соответствующего сертификата и открытого ключа. Однако Cryptoki не настаивает на такой привязке, и не требуется, чтобы сертификат и частный/секретный ключ также хранились на носителе.

Если атрибут **CKA_SENSITIVE** равен CK_TRUE или атрибут **CKA_EXTRACTABLE** равен CK_FALSE, то определенные атрибуты частного/секретного ключа не смогут открываться в незашифрованном тексте вне носителя. Указание на то, какие это атрибуты, для каждого типа личных ключей приводится в таблице атрибутов в разделе, который содержит описание данного типа ключа.

Атрибут **CKA_ALWAYS_AUTHENTICATE** может использоваться для востребования повторной аутентификации (чтобы заставить пользователя предоставить PIN) при каждом использовании частного/секретного ключа. «Использование» в данном случае означает криптографическую операцию, такую как постановка подписи или расшифрование. Данный атрибут может быть выставлен только на CK_TRUE, когда **CKA_PRIVATE** также выставлен на CK_TRUE.

Повторная аутентификация происходит при вызове **C_Login**, при этом *userType* выставляется на **CKU_CONTEXT_SPECIFIC** сразу после начала криптографической операции, использующей ключ (например, после **C_SignInit**). При таком вызове фактический тип пользователя неявно задается требованиями к использованию активного ключа. Если функция **C_Login** выдает значение CKR_OK, значит, пользователь успешно прошел аутентификацию и, таким образом, активный ключ выставляется в зарегистрированное состояние, которое продолжается до тех пор, пока не будет успешно или неудачно завершена криптографическая операция (например, через **C_Sign**, **C_SignFinal**, ...). Возвращаемое значение CKR_PIN_INCORRECT функции **C_Login** означает, что пользователю отказано в разрешении использовать данный ключ, а продолжение криптографической операции приведет к такому поведению системы, как будто функция **C_Login** и не вызывалась. В обоих этих случаях состояние сеанса остается одинаковым, однако повторные неудачные попытки пройти аутентификацию могут привести к тому, что PIN будет заблокирован. **C_Login** в данном случае выдаст сообщение CKR_PIN_LOCKED, и это также приводит к выходу пользователя из носителя. Неудачные попытки или отказ произвести повторную аутентификацию в случае, когда CKA_ALWAYS_AUTHENTICATE выставлен на CK_TRUE, приведет к тому, что после вызовов с использованием этого ключа будет выдаваться сообщение CKR_USER_NOT_LOGGED_IN. **C_Login** выдаст сообщение CKR_OPERATION_NOT_INITIALIZED, однако это не повлияет на активную криптографическую операцию, если делается попытка повторно пройти аутентификацию, а атрибут CKA_ALWAYS_AUTHENTICATE выставлен на CK_FALSE.

10.10 Объекты секретного ключа

Объекты секретного ключа (класс объекта **CKO_SECRET_KEY**) содержат секретные ключи. Таблица 31 содержит определения общих атрибутов для всех секретных ключей в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 31 – Общие атрибуты секретного ключа

Атрибут	Тип данных	Значение
CKA_SENSITIVE [8, 11]	CK_BBOOL	CK_TRUE, если объект является чувствительным (по умолчанию выставляется на CK_FALSE)
CKA_ENCRYPT [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает шифрование [9]
CKA_DECRYPT [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает расшифрование [9]
CKA_SIGN [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает электронные подписи (т. е. коды аутентификации), когда подпись является приложением к данным [9]
CKA_VERIFY [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает верификацию (т. е. коды аутентификации), когда подпись является приложением к данным [9]
CKA_WRAP [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает упаковку (т. е. может использоваться для упаковки других ключей) [9]
CKA_UNWRAP [8]	CK_BBOOL	CK_TRUE, если ключ поддерживает распаковку (т. е. может использоваться для распаковки других ключей) [9]
CKA_EXTRACTABLE [8, 12]	CK_BBOOL	CK_TRUE, если ключ является извлекаемым и может упаковываться[9]
CKA_ALWAYS_SENSITIVE [2, 4, 6]	CK_BBOOL	CK_TRUE, если атрибут ключа CKA_SENSITIVE *всегда* был выставлен на CK_TRUE
CKA_NEVER_EXTRACTABLE [2, 4, 6]	CK_BBOOL	CK_TRUE, если атрибут ключа CKA_EXTRACTABLE *никогда не* был выставлен на CK_TRUE
CKA_CHECK_VALUE	Совокупность байтов	Контрольная сумма ключа
CKA_WRAP_WITH_TRUSTED [11]	CK_BBOOL	CK_TRUE, если ключ может упаковываться только при помощи упаковочного ключа с атрибутом CKA_TRUSTED, выставленным на CK_TRUE. По умолчанию выставляется на CK_FALSE
CKA_TRUSTED [10]	CK_BBOOL	Упаковочный ключ может использоваться для упаковки ключей с использованием CKA_WRAP_WITH_TRUSTED, выставленного на CK_TRUE
CKA_WRAP_TEMPLATE	CK_ATTRIBUTE_PTR	Для упаковочных ключей. Шаблон атрибута должен совпадать с любыми ключами, упакованными с использованием упаковочного ключа. Ключи, которые не совпадают, не могут упаковываться. Количество атрибутов в массиве – это компонент *ulValueLen* атрибута, разделенный по размеру CK_ATTRIBUTE
CKA_UNWRAP_TEMPLATE	CK_ATTRIBUTE_PTR	Для упаковочных ключей. Шаблон атрибута может применяться к любым новым ключам, распакованным с использованием данного упаковочного ключа. Любой шаблон, предоставленный пользователем, применяется после этого шаблона, как если бы объект уже был создан. Количество атрибутов в массиве – это компонент *ulValueLen* атрибута, разделенный по размеру CK_ATTRIBUTE

См. сноски в таблице 15.

Если атрибут **CKA_SENSITIVE** равен CK_TRUE или атрибут **CKA_EXTRACTABLE** равен CK_FALSE, то определенные атрибуты секретного ключа не смогут открываться в открытом тексте вне носителя. Указание на то, какие это атрибуты, для каждого типа секретных ключей приводится в таблице атрибутов в разделе, который содержит описание данного типа ключа.

Атрибут контрольной суммы (Key Check Value, KCV) ключа для объектов симметричных ключей вызывается как **CKA_CHECK_VALUE,** тип – совокупность байтов (байтовый массив), длина 3 байта и действует как «отпечаток» файла или контрольная сумма ключа. Они предназначены для перекрестной проверки симметричных ключей с использованием других систем, которые совместно используют такой же ключ, а также в качестве проверки достоверности после ручного ввода с клавиатуры или восстановления из резервной копии. Смотрите определения объектов специфических видов ключей для алгоритмов KCV.

Свойства:

1. Для двух ключей, которые криптографически идентичны, величина данного атрибута должна быть одинаковой.

2. Атрибут CKA_CHECK_VALUE не должен быть пригоден для получения любой части значения ключа.

3. Неуникальность. Два разных ключа могут иметь одно и то же значение CKA_CHECK_VALUE. Это маловероятно, но возможно. Вероятность легко вычисляется.

Данные атрибуты являются необязательными, но, если они поддерживаются, то значение атрибута всегда предоставляется библиотекой, независимо от того, как создавался или выводился объект ключа. Это значение должно предоставляться, даже если операция шифрования для данного ключа запрещена (т. е., когда CKA_ENCRYPT выставлен на CK_FALSE).

Если значение предоставляется в шаблоне приложения (разрешается, но абсолютно необязательно), то, если оно поддерживается, оно должно совпадать со значением, вычисленным для данного атрибута библиотекой, или библиотека выдаст сообщение CKR_ATTRIBUTE_VALUE_INVALID. Если библиотека не поддерживает данный атрибут, она должна его игнорировать. Такое отношение к атрибуту из шаблона не навредит процессу и позволит рассматривать данный атрибут, как и любой другой атрибут при упаковке и распаковке ключей, где атрибуты также сохраняются.

Генерации KCV может помешать приложение, если оно предоставит атрибут в шаблоне в качестве записи без значения (длина 0). Приложение может запросить это значение в любое время, как и любой другой атрибут с использованием C_GetAttributeValue. C_SetAttributeValue может использоваться для уничтожения атрибута путем наделения его пустым значением.

Если иное не указано в определении объекта, значение данного атрибута выводится из объекта ключа, для чего берутся три первых байта одного зашифрованного блока нулевых (0x00) байтов, с использованием шифра по умолчанию и режима (например, электронной кодовой книги), связанного с типом ключа объекта секретного ключа.

10.11 Объекты параметра домена

10.11.1 Определения

В настоящем разделе определяется класс объекта CKO_DOMAIN_PARAMETERS как объект типа CK_OBJECT_CLASS, используемый в атрибуте объектов CKA_CLASS.

10.11.2 Обзор

Данный класс объектов создан для поддержки хранения расширенных параметров определенного алгоритма. Как DSA, так и DH используют параметры домена на стадии генерации пары ключей. В частности, некоторые библиотеки поддерживают генерацию параметров домена (изначально вне сферы действия PKCS11), поэтому и был добавлен этот класс объектов.

Чтобы использовать объект параметра домена, вы должны извлечь атрибуты в шаблон и предоставить (все еще в шаблоне) их соответствующей функции генерации пары ключей.

Объекты параметра домена (класс объекта **CKO_DOMAIN_PARAMETERS**) содержат параметры общего домена.

Таблица 32 содержит определения общих атрибутов для всех объектов параметров домена в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 32 – Общие атрибуты параметра домена

Атрибут	Тип данных	Значение
CKA_KEY_TYPE [1]	CK_KEY_TYPE	Тип ключа, для генерации которого могут использоваться данные параметры домена
CKA_LOCAL [2, 4]	CK_BBOOL	CK_TRUE, только если соблюдается одно из следующих условий: • параметры домена были сгенерированы локально (т. е. на носителе) при помощи вызова функций **C_GenerateKey** • параметры домена были созданы при помощи вызова функции **C_CopyObject** как копия параметров домена, атрибут **CKA_LOCAL** которых был выставлен на CK_TRUE
См. сноски в таблице 15.		

Атрибут **CKA_LOCAL** имеет значение CK_TRUE тогда и только тогда, когда значения параметров домена были изначально сгенерированы на носителе с использованием вызова функции **C_GenerateKey.**

10.12 Объекты механизма

10.12.1 Определения

В настоящем разделе определяется класс объекта CKO_MECHANISM как объект типа CK_OBJECT_CLASS, используемый в атрибуте объектов CKA_CLASS.

10.12.2 Обзор

Объекты механизма предоставляют информацию о механизмах, поддерживаемых устройством, кроме тех, которые выдаются структурой **CK_MECHANISM_INFO**.

При поиске объектов с использованием **C_FindObjectsInit** и **C_FindObjects** объекты механизмов не будут выдаваться, если атрибут **CKA_CLASS** в шаблоне не имеет значение **CKO_MECHANISM**. Это предотвращает возможность того, что приложения, написанные для предыдущих версий Cryptoki, будут находить непонятные для них объекты.

Таблица 33 – Общие атрибуты механизма

Атрибут	Тип данных	Значение
CKA_MECHANISM_TYPE	CK_MECHANISM_TYPE	Тип объекта механизма

Атрибут **CKA_MECHANISM_TYPE** может быть не выставлен.

11 Функции

Функции Cryptoki организованы в следующие категории:
- функции общего назначения (4 функции);
- функции управления слотом и носителем (9 функций);
- функции управления сеансом (8 функций);
- функции управления объектами (9 функций);
- функции шифрования (4 функции);
- функции расшифрования (4 функции);
- функции списка сообщения (5 функций);
- функции электронной подписи и MAC (6 функций);
- функции верификации электронных подписей и MAC (6 функций);
- криптографические функции двойного назначения (4 функции);
- функции управления ключами (5 функций);
- функции генерации случайных чисел (2 функции);
- функции управления параллельными функциями (2 функции).

В дополнение к этим функциям Cryptoki может использовать предоставляемые приложением функции обратного вызова для уведомления приложения об определенных событиях и может также использовать предоставляемые приложением функции для управления объектами-флагами для безопасного многопоточного доступа к библиотеке.

Выполнение вызовов функций Cryptoki в целом осуществляется по принципу «все или ничего», т. е. вызов функции либо полностью выполняет поставленную перед ним задачу, либо не выполняет вообще ничего.

- Если функция Cryptoki выполняется успешно, она выдает значение CKR_OK.
- Если функция Cryptoki не выполняется успешно, она выдает некоторое значение, отличающееся от CKR_OK, а носитель находится в состоянии, которое предшествовало вызову функции. Если предполагалось, что вызов функции изменит содержание определенных участков памяти на локальном компьютере, то эти участки памяти могут все же быть изменены, несмотря на отказ функции.
- При необычных (крайне неприятных) обстоятельствах функция может завершиться отказом с выдачей сообщения CKR_GENERAL_ERROR. Когда такое происходит, носитель и/или локальный компьютер может находиться в неустойчивом состоянии, при этом цели функции могут быть частично достигнуты.

Существует небольшое количество функций, чьи возвращаемые значения ведут себя не вполне так, как описано выше; эти исключения описаны индивидуально вместе с описанием самих функций.

Библиотека Cryptoki не должна поддерживать каждую функцию в Cryptoki API. Однако даже у неподдерживаемой функции должно быть «ответвление» в библиотеке, которое просто выдает значение CKR_FUNCTION_NOT_SUPPORTED. Запись функции в структуре библиотеки **CK_FUNCTION_LIST** (согласно информации, получаемой при помощи функции **C_GetFunctionList**) должна указывать на это «ответвление» функции (см. 9.6).

11.1 Возвращаемые значения функции

Интерфейс Cryptoki располагает большим количеством функций и возвращаемых значений. В 11.1 мы перечисляем различные возможные возвращаемые значения функций Cryptoki; большая часть оставшейся информации из 10.12 посвящена подробному описанию поведения функций Cryptoki, включая информацию о том, какие возвращаемые величины могут выдаваться каждой из этих функций.

В связи со сложностью спецификации Cryptoki рекомендуется, чтобы приложения Cryptoki пытались давать некоторую свободу действий при интерпретации возвращаемых величин функций Cryptoki. Мы попытались описать поведение функций Cryptoki как можно более полно, тем не менее очевидно, что в описании есть некоторые пробелы. Например, возможно, что конкретный код ошибки, который может применяться к конкретной функции Cryptoki, к сожалению, фактически не приведен в числе возможных кодов ошибки для данной функции. Возможен вариант, при котором разработчик библиотеки Cryptoki все же сможет разрешить в своей реализации данной функции возврат данного кода ошибки. Естественно, внезапный аварийный сброс приложения, использующего данную библиотеку, при получении такого кода ошибки для этой функции выглядел бы некрасиво. Гораздо более предпочтительным был бы вариант, при котором приложение исследовало бы возвращаемое значение функции, определило бы, что это значение указывает на определенного рода ошибку (даже если приложение не знало бы точно, *какая* это ошибка), и действовало бы в дальнейшем соответственно.

См. 11.1.8 для получения более подробной информации о том, как разработчик может попытаться создать приложение, которое обеспечивало бы нормальную работу при разнообразном поведении библиотек Cryptoki.

11.1.1 Универсальные возвращаемые значения функций Cryptoki

Любая функция Cryptoki может возвращать любое из следующих значений:

- CKR_GENERAL_ERROR: Произошла непоправимая ошибка. В наихудшем случае возможно, что функция могла только частично завершиться успешно и что компьютер и/или носитель находятся в неустойчивом состоянии;
- CKR_HOST_MEMORY: Компьютер, на котором работает библиотека Cryptoki, не имеет достаточно памяти для выполнения запрашиваемой функции;
- CKR_FUNCTION_FAILED: Запрашиваемая функция не может быть выполнена, однако подробная информации о том, почему она не может быть выполнена, не предоставляется в данном сообщении с кодом ошибки. Если неудачно завершившаяся функция использует сеанс, возможно, что структура **CK_SESSION_INFO**, которую можно получить при помощи вызова **C_GetSessionInfo**, будет содержать полезную информацию о том, что произошло в ее поле *ulDeviceError*. Однако в случае, если вызов функции завершился неудачно, не обязательно, что ситуация совсем безнадежна, такова, какая была бы в случае, если был бы выдан код CKR_GENERAL_ERROR. В зависимости от того, что фактически являлось основной причиной ошибки, возможно, что попытка выполнить тот же самый вызов функции снова будет удачной;

• CKR_OK: Функция выполнена успешно. С технической точки зрения CKR_OK – это *не совсем* «универсальное» возвращаемое значение; в частности, унаследованные функции **C_GetFunctionStatus** и **C_CancelFunction** (см. 11.16) не могут выдавать CKR_OK.

Список относительных приоритетов данных ошибок приводился выше в соответствующем порядке, например, если CKR_GENERAL_ERROR или CKR_HOST_MEMORY будут являться несоответствующими возвращаемыми кодами ошибок, то должен выдаваться код CKR_GENERAL_ERROR.

11.1.2 Возвращаемые значения функций Cryptoki для функций, которые используют идентификаторы сеансов

Любая функция Cryptoki, которая принимает идентификатор сеанса в качестве одного из своих аргументов (т. е. любая функция Cryptoki, за исключением **C_Initialize**, **C_Finalize**, **C_GetInfo**, **C_GetFunctionList**, **C_GetSlotList**, **C_GetSlotInfo**, **C_GetTokenInfo**, **C_WaitForSlotEvent**, **C_GetMechanismList**, **C_GetMechanismInfo**, **C_InitToken**, **C_OpenSession** и **C_CloseAllSessions**), может выдать следующие значения:

• CKR_SESSION_HANDLE_INVALID: Указанный идентификатор сеанса был недействительным *в момент вызова функции.* Это может произойти, если носитель сеанса удален до запуска функции, так как удаление носителя закрывает все его сеансы;

• CKR_DEVICE_REMOVED: Носитель был удален из его слота *во время выполнения функции;*

• CKR_SESSION_CLOSED: Сеанс был закрыт *во время выполнения функции.* Согласно 6.7.6 поведение Cryptoki *не задано* на случай, когда несколько потоков приложения одновременно пытаются получить доступ к общему сеансу Cryptoki. Следовательно, фактически нет гарантий того, что вызов функции когда-нибудь выдаст значение CKR_SESSION_CLOSED, если один поток использует сеанс, когда другой поток закрывает его, − пример того, как несколько потоков пытаются получить доступ к общим сеансам одновременно.

Список относительных приоритетов данных ошибок приводился выше в соответствующем порядке, *например,* если CKR_SESSION_HANDLE_INVALID или CKR_DEVICE_REMOVED будут являться несоответствующими возвращаемыми кодами ошибок, то должен возвращаться код CKR_SESSION_ HANDLE_INVALID.

На практике зачастую не столь важно (или не всегда возможно), чтобы библиотека Cryptoki могла делать различие между вариантами, когда носитель удаляется *до* вызова функции и когда носитель удаляется *во время* выполнения функции.

11.1.3 Возвращаемые значения функций Cryptoki для функций, которые используют носитель

Любая функция Cryptoki, которая использует определенный носитель (т. е. функции Cryptoki, за исключением **C_Initialize**, **C_Finalize**, **C_GetInfo**, **C_GetFunctionList**, **C_GetSlotList**, **C_GetSlotInfo** или **C_WaitForSlotEvent**), может выдать любое из следующих значений:

• CKR_DEVICE_MEMORY: Носитель не располагает достаточным количеством памяти для выполнения запрашиваемой функции;

• CKR_DEVICE_ERROR: Возникли какие-то проблемы с носителем и/или слотом. Данный код ошибки может выдаваться не только функциями, перечисленными выше; в частности, возможен вариант, при котором значение CKR_DEVICE_ERROR будет выдано функцией **C_GetSlotInfo**;

• CKR_TOKEN_NOT_PRESENT: Носитель отсутствовал в слоте *во время запуска функции*;

• CKR_DEVICE_REMOVED: Носитель был удален из слота *во время выполнения функции.*

Относительные приоритеты данных ошибок соответствуют порядку, перечисленному выше, *например,* если правильным кодом ошибки является либо CKR_DEVICE_MEMORY, либо CKR_DEVICE_ERROR, то выдаваться должен код CKR_DEVICE_MEMORY.

На практике зачастую не столь важно (или не всегда возможно), чтобы библиотека Cryptoki могла делать различие между фактом удаления носителя *до* вызова функции и фактом удаления носителя *во время* выполнения функции.

11.1.4 Специальное возвращаемое значение для обратных вызовов, предоставляемых приложением

Существует специализированное возвращаемое значение, которое выдается любой функцией в описываемом Cryptoki API, однако оно же может быть возвращаемым значением функции возврата, предоставленной приложением. Это значение:

• CKR_CANCEL: Когда функция, выполняемая вместе с приложением, принимает решение предоставить приложению возможность выполнить определенную работу, она вызывает функцию, предоставляемую приложением при помощи отзыва CKN_SURRENDER (см. 11.17). Если в результате этого отзыва выдается значение CKR_CANCEL, то функция прерывается и выдает значение

CKR_FUNCTION_CANCELED.

11.1.5 Специальные возвращаемые значения для функций управления флагами

Существует два других специализированных значения функции, которые не выдаются ни одной из функций описываемого Cryptoki. Эти значения могут выдаваться функциями управления флагами, которые предоставляются приложением, и могут спокойно игнорироваться разработчиками приложения, которые не используют свою собственную потоковую модель. Это значения:

- CKR_MUTEX_BAD: Данный код ошибки может выдаваться функциями управления флагами, которым в качестве аргумента предоставляется флаг с ошибкой. К сожалению, возможен вариант, при котором такая функция не распознает флаг с ошибкой. Следовательно, нет гарантии, что такая функция будет удачно распознавать флаги с ошибками и возвращать данное значение;
- CKR_MUTEX_NOT_LOCKED: Данный код ошибки может возвращаться функциями разблокировки флагов. Он указывает на то, что флаг, предоставленный для обработки функцией разблокировки флага, не был заблокирован.

11.1.6 Все остальные возвращаемые значения функций Cryptoki

Ниже приводится описание других возвращаемых значений функций Cryptoki. За исключением того, что упоминалось в описаниях определенных кодов ошибок, вообще не существует определенных приоритетов среди ошибок, перечисленных ниже, т. е. если к процессу исполнения функции может применяться более чем один код ошибки, то функция может выдать любой из подходящих кодов.

- CKR_ARGUMENTS_BAD: Это, скорее, общий код ошибки, который указывает, что аргументы, предоставляемые для обработки функцией Cryptoki, по той или иной причине являются неподходящими.
- CKR_ATTRIBUTE_READ_ONLY: Была сделана попытка задать значение атрибута, которое не может задаваться приложением или не может изменяться приложением. Для получения более подробной информации см. 10.1.
- CKR_ATTRIBUTE_SENSITIVE: Была совершена попытка получить значение атрибута объекта, которая не может быть удовлетворена в связи с тем, что объект является либо уязвимым, либо неизвлекаемым.
- CKR_ATTRIBUTE_TYPE_INVALID: В шаблоне указан недействительный тип атрибута. Для получения более подробной информации см. 10.1.
- CKR_ATTRIBUTE_VALUE_INVALID: Для определенного атрибута в шаблоне указано недействительное значение. Для получения более подробной информации см. 10.1.
- CKR_BUFFER_TOO_SMALL: Выходные данные функции слишком велики, чтобы поместиться в предоставленном буфере.
- CKR_CANT_LOCK: Данное значение может выдаваться только функцией **C_Initialize**. Оно означает, что тип блокировки, запрошенный приложением для безопасности потока, в данной библиотеке недоступен, и поэтому приложение не может использовать библиотеку указанным способом.
- CKR_CRYPTOKI_ALREADY_INITIALIZED: Данное значение может выдаваться только функцией **C_Initialize**. Это означает, что библиотека Cryptoki уже была инициализирована (при предыдущем вызове **C_Initialize**, которому не сопутствовал соответствующий вызов функции **C_Finalize**).
- CKR_CRYPTOKI_NOT_INITIALIZED: Данное значение может выдаваться любой функцией, кроме **C_Initialize** и **C_GetFunctionList**. Оно указывает на невозможность выполнения функции в связи с тем, что библиотека Cryptoki еще не инициализирована запросом к **C_Initialize**.
- CKR_DATA_INVALID: Нешифрованные входные данные для криптографической операции являются недействительными. Приоритет данного возвращаемого значения ниже, чем приоритет значения CKR_DATA_LEN_RANGE.
- CKR_DATA_LEN_RANGE: Нешифрованные входные данные для криптографической операции имеют неверную длину. В зависимости от механизма операции это может означать, что нешифрованные данные являются слишком короткими или слишком длинными или что их длина не кратна размеру конкретного блока. Приоритет данного возвращаемого значения выше, чем приоритет значения CKR_DATA_INVALID.
- CKR_DOMAIN_PARAMS_INVALID: Функции предоставлены недействительные или неподдерживаемые параметры домена. Поддерживаемые данным механизмом методы представления параметров домена могут варьироваться от носителя к носителю.
- CKR_ENCRYPTED_DATA_INVALID: Шифрованные входные данные для криптографической операции определены как неверный шифртекст. Приоритет данного возвращаемого значения ниже, чем приоритет значения CKR_ENCRYPTED_DATA_LEN_RANGE.
- CKR_ENCRYPTED_DATA_LEN_RANGE: Шифрованные входные данные для криптографиче-

ской операции определены как неверный шифртекст исключительно по причине их длины. В зависимости от механизма операции это может означать, что шифртекст является слишком коротким или слишком длинным либо что его длина не кратна размеру конкретного блока. Приоритет данного возвращаемого значения выше, чем приоритет значения CKR_ENCRYPTED_DATA_INVALID.

- CKR_FUNCTION_CANCELED: Функция была отменена в процессе исполнения. Это происходит с криптографической функцией, если она осуществляет отзыв приложения **CKN_SURRENDER**, в результате чего выдается значение CKR_CANCEL (см. CKR_CANCEL). То же происходит и с функцией, которая осуществляет ввод PIN через защищенный путь. Способ, используемый для отмены ввода PIN по защищенному пути, зависит от устройства.
- CKR_FUNCTION_NOT_PARALLEL: В данный момент не существует функции, исполняемой параллельно в указанном сеансе. Это наследуемый код ошибки, который выдается только функциями наследования **C_GetFunctionStatus** и **C_CancelFunction**.
- CKR_FUNCTION_NOT_SUPPORTED: Запрошенная функция не поддерживается данной библиотекой Cryptoki. Даже неподдерживаемые в Cryptoki API функции должны иметь «ответвление» в данной библиотеке; это ответвление должно просто выдавать CKR_FUNCTION_NOT_SUPPORTED.
- CKR_FUNCTION_REJECTED: Запрос подписи отклонен пользователем.
- CKR_INFORMATION_SENSITIVE: Запрошенная информация не может быть получена, поскольку носитель считает ее уязвимой и не может или не желает открывать ее.
- CKR_KEY_CHANGED: Данное значение выдается только функцией **C_SetOperationState**. Оно указывает, что один из заданных ключей не является тем ключом, который использовался в изначально сохраненном сеансе.
- CKR_KEY_FUNCTION_NOT_PERMITTED: Сделана попытка использовать ключ в криптографических целях, которые не разрешены, исходя из атрибутов ключа. Например, чтобы использовать ключ для шифрования, в этом ключе атрибут **CKA_ENCRYPT** должен быть выставлен на CK_TRUE (тот факт, что у ключа должен быть атрибут **CKA_ENCRYPT**, предполагает, что ключ не должен быть секретным). Приоритет данного возвращаемого значения ниже, чем приоритет значения CKR_KEY_TYPE_INCONSISTENT.
- CKR_KEY_HANDLE_INVALID: Заданный идентификатор ключа недействителен. Может быть так, что указанный идентификатор будет действительным для объекта, не являющегося ключом. Повторим, что 0 никогда не является действительным идентификатором ключа.
- CKR_KEY_INDIGESTIBLE: Данный код ошибки может выдаваться только функцией **C_DigestKey**. Он указывает, что значение указанного ключа не может быть подвергнуто функции списка по каким-либо причинам (возможно, ключ не является секретным или носитель просто не может вычислить список такого вида ключа).
- CKR_KEY_NEEDED: Данное значение выдается только функцией **C_SetOperationState**. Оно указывает, что состояние сеанса не может быть восстановлено, так как функции **C_SetOperationState** необходимо предоставить один или более ключей, которые использовались в изначально сохраненном сеансе.
- CKR_KEY_NOT_NEEDED: Функции **C_SetOperationState** предоставлен посторонний ключ. Например, предпринимается попытка восстановления сеанса, который выполнял операцию получения списка, а предоставленный ключ является ключом шифрования.
- CKR_KEY_NOT_WRAPPABLE: Хотя в заданном частном/секретном или секретном ключе атрибут CKA_UNEXTRACTABLE не выставлен на CK_TRUE, Cryptoki (или сам носитель) не может упаковать ключ в соответствии с поступившим запросом (возможно, носитель может упаковать данный ключ только с использованием определенных типов ключей, а указанный упаковочный ключ не относится к данным типам). Сравните с CKR_KEY_UNEXTRACTABLE.
- CKR_KEY_SIZE_RANGE: Хотя запрошенные криптографические операции, защищенные ключами, в принципе, могут быть выполнены, данная библиотека Cryptoki (или носитель) фактически не может сделать этого, так как длина предоставленного ключа не входит в диапазон размеров ключа, с которыми она может работать.
- CKR_KEY_TYPE_INCONSISTENT: Тип указанного ключа не является верным для использования с указанным механизмом. Приоритет данного возвращаемого значения выше, чем приоритет значения CKR_KEY_FUNCTION_NOT_PERMITTED.
- CKR_KEY_UNEXTRACTABLE: Указанный частный/секретный или секретный ключ не может быть упакован, так как его атрибут CKA_UNEXTRACTABLE выставлен на CK_TRUE. Сравните с CKR_KEY_NOT_WRAPPABLE.

- CKR_MECHANISM_INVALID: Для криптографической операции указан неверный механизм. Данный код ошибки является правильным возвращаемым значением, если был указан неизвестный механизм или если указанный механизм не может быть использован в выбранном носителе с выбранной функцией.
- CKR_MECHANISM_PARAM_INVALID: Механизму, назначенному для криптографической операции, предоставлены недействительные параметры. Значения параметров, поддерживаемые определенным механизмом, варьируются от носителя к носителю.
- CKR_NEED_TO_CREATE_THREADS: Данное значение может выдаваться только функцией **C_Initialize**. Оно выдается при соблюдении двух условий:

1. Приложение вызвало функцию **C_Initialize** способом, указывающим библиотеке Cryptoki, что потоки, выполняющие запросы к библиотеке, не могут использовать методики собственной операционной системы для порождения новых потоков;
2. Библиотека не может работать надлежащим образом без возможности порождать новые потоки вышеуказанным способом.

- CKR_NO_EVENT: Данное значение может быть выдано только функцией **C_GetSlotEvent**. Оно выдается, когда **C_GetSlotEvent** вызывается в режиме отсутствия блокировки и при этом на слоте нет новых событий, порождающих возвращаемые значения.
- CKR_OBJECT_HANDLE_INVALID: Указанный идентификатор объекта не является действительным. Повторим, что 0 никогда не является действительным идентификатором объекта.
- CKR_OPERATION_ACTIVE: Уже есть активная операция (или комбинация активных операций), которая не дает Cryptoki активизировать указанную операцию (например, активная операция поиска объекта не позволит Cryptoki активизировать операцию шифрования с использованием **C_EncryptInit**, или активная операция получения списка и активная операция шифрования не позволят Cryptoki активизировать операцию постановки электронной подписи, или на носителе, который не поддерживает одновременные двойные криптографические операции в одном сеансе (см. описание флага **CKF_DUAL_CRYPTO_OPERATIONS** в структуре **CK_TOKEN_INFO**), активная операция постановки электронной подписи не позволит Cryptoki активизировать операцию шифрования).
- CKR_OPERATION_NOT_INITIALIZED: Нет активной операции соответствующего типа в указанном сеансе. Например, приложение не может вызвать функцию **C_Encrypt** в сеансе, если до этого оно сначала не вызвало **C_EncryptInit**, чтобы активизировать операцию шифрования.
- CKR_PIN_EXPIRED: Время использования указанного PIN истекло, а запрошенная операция не может выполняться, если не была вызвана функция C_SetPIN для изменения значения PIN. Наличие или отсутствие вероятности истечения действия PIN на носителе варьируется от носителя к носителю.
- CKR_PIN_INCORRECT: Указанный PIN неверен, т. е. он не совпадает со значением PIN, хранящимся на носителе. В более общем случае, когда в аутентификацию на носителе входит не PIN, а что-либо другое, – попытка пройти аутентификацию не удалась.
- CKR_PIN_INVALID: Указанный PIN содержит недействительные символы. Данный возвращаемый код применяется только к функциям, которые пытаются задать PIN.
- CKR_PIN_LEN_RANGE: Указанный PIN является слишком длинным или слишком коротким. Данный возвращаемый код применяется только к функциям, которые пытаются задать PIN.
- CKR_PIN_LOCKED: Указанный PIN «заблокирован» и не может использоваться. Это означает, что в связи с достижением определенного количества неудачных попыток пройти аутентификацию носитель не желает разрешать дальнейшие попытки аутентификации. В зависимости от носителя указанный PIN может оставаться, а может не оставаться заблокированным на неопределенный срок.
- CKR_RANDOM_NO_RNG: Данное значение может выдаваться функциями **C_SeedRandom** и **C_GenerateRandom**. Оно означает, что указанный носитель не содержит генератора случайных чисел. Приоритет данного возвращаемого значения выше, чем приоритет значения CKR_RANDOM_ SEED_NOT_SUPPORTED.
- CKR_RANDOM_SEED_NOT_SUPPORTED: Данное значение может выдаваться только функцией **C_SeedRandom**. Оно указывает, что генератор случайных чисел не принимает посев чисел из приложения. Приоритет данного возвращаемого значения ниже, чем приоритет значения CKR_RANDOM_ NO_RNG.
- CKR_SAVED_STATE_INVALID: Данное значение может выдаваться только функцией **C_SetOperationState**. Оно указывает, что предоставленное сохраненное состояние криптографических операций недействительно, и поэтому не может быть восстановлено в конкретном сеансе.

- CKR_SESSION_COUNT: Данное значение может выдаваться только функцией

C_OpenSession. Оно указывает, что попытка открыть сеанс не удалась либо в связи с тем, что у носителя уже открыто слишком много сеансов, либо потому, что у носителя уже открыто слишком много сеансов чтения/записи.

- CKR_SESSION_EXISTS: Данное значение может выдаваться только функцией **C_InitToken**. Оно указывает, что уже открыт сеанс с носителем, и поэтому носитель невозможно инициализировать.
- CKR_SESSION_PARALLEL_NOT_SUPPORTED: Указанный носитель не поддерживает параллельные сеансы. Это унаследованный код ошибки, так как в Cryptoki версии 2.01 и выше *ни один* носитель не поддерживает параллельные сеансы. CKR_SESSION_PARALLEL_NOT_SUPPORTED может выдаваться только функцией **C_OpenSession** и только тогда, когда **C_OpenSession** вызывается особым (отмененным) способом.
- CKR_SESSION_READ_ONLY: Указанный сеанс не смог выполнить желаемое действие, так как он является сеансом только для чтения. Приоритет данного возвращаемого значения ниже, чем приоритет значения CKR_TOKEN_WRITE_PROTECTED.
- CKR_SESSION_READ_ONLY_EXISTS: Сеанс только для чтения уже существует, поэтому ответственный за безопасность (SO) не может зарегистрироваться.
- CKR_SESSION_READ_WRITE_SO_EXISTS: Уже существует сеанс чтения/записи ответственного за безопасность (SO), поэтому невозможно открыть сеанс только для чтения.
- CKR_SIGNATURE_LEN_RANGE: Предоставленные электронная подпись/MAC могут рассматриваться как недействительные, исходя исключительно из их длины. Приоритет данного возвращаемого значения выше, чем приоритет значения CKR_SIGNATURE_INVALID.
- CKR_SIGNATURE_INVALID: Предоставленные электронная подпись/MAC являются недействительными. Приоритет данного возвращаемого значения ниже, чем приоритет значения CKR_SIGNATURE_ LEN_RANGE.
- CKR_SLOT_ID_INVALID: Указанный ID слота является недействительным.
- CKR_STATE_UNSAVEABLE: Состояние криптографических операций указанного сеанса не может быть сохранено по какой-либо причине (возможно, носитель просто не может сохранить текущее состояние). Приоритет данного возвращаемого значения ниже, чем приоритет значения CKR_OPERATION_ NOT_INITIALIZED.
- CKR_TEMPLATE_INCOMPLETE: Шаблон, указанный для создания объекта, является неполным или в нем недостает некоторых необходимых атрибутов. Для получения более подробной инфор-мации см. 10.1.
- CKR_TEMPLATE_INCONSISTENT: Шаблон, указанный для создания объекта, содержит конфликтующие атрибуты. Для получения более подробной информации см. 10.1.
- CKR_TOKEN_NOT_RECOGNIZED: Библиотека Cryptoki и/или слот не распознает носитель в слоте.
- CKR_TOKEN_WRITE_PROTECTED: Запрошенное действие не может быть выполнено в связи с тем, что носитель защищен от записи. Приоритет данного возвращаемого значения выше, чем приоритет значения CKR_SESSION_READ_ONLY.
- CKR_UNWRAPPING_KEY_HANDLE_INVALID: Данное значение может выдаваться только функцией **C_UnwrapKey**. Оно указывает, что идентификатор ключа, назначенного к использованию для распаковки другого ключа, является недействительным.
- CKR_UNWRAPPING_KEY_SIZE_RANGE: Данное значение может выдаваться только функцией **C_UnwrapKey**. Оно указывает на то, что, хотя запрошенная операция распаковки, в принципе, может быть выполнена, данная библиотека Cryptoki (или сам носитель) фактически не может сделать этого, так как длина предоставленного ключа не входит в диапазон размеров ключей, с которыми она может работать.
- CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT: Данное значение может выдаваться только функцией **C_UnwrapKey**. Оно указывает, что тип ключа, назначенного к использованию для распаковки другого ключа, противоречит механизму, указанному для распаковки.
- CKR_USER_ALREADY_LOGGED_IN: Данное значение может выдаваться только функцией **C_Login**. Оно указывает, что данный пользователь не может войти в сеанс, так как он уже зарегистрирован в нем. Например, если приложение имеет открытый сеанс SO и пытается зарегистрировать в нем SO, оно получит сообщение с данным кодом ошибки.
- CKR_USER_ANOTHER_ALREADY_LOGGED_IN: Данное значение может выдаваться только функцией **C_Login**. Оно указывает, что данный пользователь не может войти в сеанс, так как другой пользователь уже зарегистрирован в нем. Например, если приложение имеет открытый сеанс SO и пытается зарегистрировать в нем обычного пользователя, оно получит сообщение с данным кодом ошибки.

• CKR_USER_NOT_LOGGED_IN: Желаемое действие не может быть совершено, так как соответствующий пользователь (или *любой* подходящий пользователь) не зарегистрирован. Одним из примеров является ситуация, когда из сеанса нельзя выйти, если этот сеанс не зарегистрирован. Еще один
пример – это, когда невозможно создать секретный объект на носителе, если сеанс, который пытается создать этот объект, не зарегистрирован как обычный пользователь. Последний пример – ситуация, когда невозможно выполнить криптографические операции на определенных носителях, если обычный пользователь не зарегистрирован.

• CKR_USER_PIN_NOT_INITIALIZED: Данное значение может выдаваться только функцией **C_Login**. Оно указывает, что PIN обычного пользователя еще не инициализирован функцией **C_InitPIN**.

• CKR_USER_TOO_MANY_TYPES: Произведена попытка одновременно зарегистрировать на носителе больше различных пользователей, чем это разрешает носитель и/или библиотека. Например, если какое-то приложение имеет открытый сеанс SO, а другое приложение пытается зарегистрировать в сеансе обычного пользователя, попытка, возможно, завершится выдачей сообщения с данным кодом ошибки. Однако обязательного требования об этом нет. **C_Login** должна выдавать это значение, только если не может быть поддержана одновременная работа нескольких различных пользователей. Данный код ошибки является общим для носителей, реально поддерживающих работу с несколькими пользователями.

• CKR_USER_TYPE_INVALID: За недействительным значением закреплен код **CK_USER_TYPE**. Действительными типами являются **CKU_SO**, **CKU_USER** и **CKU_CONTEXT_SPECIFIC**.

• CKR_WRAPPED_KEY_INVALID: Данное значение может выдаваться только функцией **C_UnwrapKey**. Оно указывает, что предоставленный упакованный ключ недействителен. Если к функции **C_UnwrapKey** делается вызов на распаковку определенного типа ключа (т. е. некоторый определенный тип ключа указан в шаблоне, предоставленном функции **C_UnwrapKey**), а предоставленный функции **C_UnwrapKey** упакованный ключ распознается как не являющийся упакованным ключом надлежащего типа, функция **C_UnwrapKey** должна выдавать CKR_WRAPPED_KEY_INVALID. Приоритет данного возвращаемого значения ниже, чем приоритет значения CKR_WRAPPED_KEY_LEN_RANGE.

• CKR_WRAPPED_KEY_LEN_RANGE: Данное значение может выдаваться только функцией **C_UnwrapKey**. Оно указывает, что предоставленный упакованный ключ рассматривается как недействительный, исходя исключительно из его длины. Приоритет данного возвращаемого значения выше, чем приоритет значения CKR_WRAPPED_KEY_INVALID.

• CKR_WRAPPING_KEY_HANDLE_INVALID: Данное значение может выдаваться только функцией **C_WrapKey**. Оно указывает, что идентификатор ключа, назначенного для упаковки другого ключа, является недействительным.

• CKR_WRAPPING_KEY_SIZE_RANGE: Данное значение может выдаваться только функцией **C_WrapKey**. Оно указывает на то, что, хотя запрошенная операция упаковки в принципе может быть выполнена, данная библиотека Cryptoki (или сам носитель) фактически не может сделать этого, так как длина предоставленного упаковочного ключа не входит в диапазон размеров ключей, с которыми она может работать.

CKR_WRAPPING_KEY_TYPE_INCONSISTENT: Данное значение может выдаваться только функцией **C_WrapKey**. Оно указывает, что тип ключа, назначенного для упаковки другого ключа, противоречит механизму, указанному для упаковки.

11.1.7 Дополнительная информация по относительным приоритетам ошибок Cryptoki

В общем случае, когда делается вызов, коды ошибок из 11.1.1 (а не CKR_OK) являются приоритетными по отношению к кодам ошибок из 11.1.2, которые в свою очередь имеют приоритет по отношению к кодам ошибок из 11.1.3, а те в свою очередь имеют приоритет по отношению к кодам ошибок из 11.1.6. Следует обратить внимание, что функции, использующие идентификаторы сеансов (т. е. *большинство* функций), никогда не выдают кода ошибки CKR_TOKEN_NOT_PRESENT (они выдают CKR_SESSION_HANDLE_INVALID). В дополнение к перечисленным приоритетам необходимо отметить, что если к результату вызова Cryptoki подходит более одного кода ошибки, то выдан может быть любой из них. Исключения из этого правила будут подробно перечислены в описаниях функций.

11.1.8 Сбои кода ошибки

Вот краткий список некоторых особенностей, касающихся возвращаемых величин, о которых разработчикам Cryptoki желательно было бы знать:

1. Как упомянуто в 11.1.2 и 11.1.3, библиотека Cryptoki может не обладать способностью делать

различие между извлечением носителя *до* или *во время* вызова функции;

2. Как упомянуто в 11.1.2, приложение никогда не должно рассчитывать на получение кода ошибки CKR_SESSION_CLOSED;

3. Разница между CKR_DATA_INVALID и CKR_DATA_LEN_RANGE может быть довольно тонкой. Если приложению *не нужна* способность делать различие между этими возвращаемыми значениями, наилучший вариант – рассматривать их равнозначно;

4. Аналогично, различия между CKR_ENCRYPTED_DATA_INVALID и CKR_ENCRYPTED_DATA_LEN_RANGE, а также между CKR_WRAPPED_KEY_INVALID и CKR_WRAPPED_KEY_LEN_RANGE могут быть тонкими и, возможно, наилучший вариант – рассматривать эти возвращаемые значения равнозначно;

5. Даже руководствуясь рекомендациями из 10.1, разработчику библиотеки Cryptoki может быть сложно определить, какой из кодов должен быть выдан – CKR_ATTRIBUTE_VALUE_INVALID, CKR_TEMPLATE_INCOMPLETE или CKR_TEMPLATE_INCONSISTENT. Разработчикам приложения рекомендуется по возможности свободно интерпретировать эти коды ошибок.

11.2 Принятые нормы для функций, которые выдают выходные данные в буферах переменной длины

Некоторые функции, описанные в Cryptoki, выдают выходные данные, которые производятся определенным криптографическим механизмом. Те выходные данные, которые возвращаются этими функциями, выдаются в предоставляемом приложением буфере переменной длины. Примером функции такого рода может служить **C_Encrypt**, которая в качестве аргумента принимает некоторый открытый текст и выдает буфер, заполненный шифртекстом.

Для этих функций есть некоторые общие нормы вызова, описание которых здесь приводится. Два из аргументов данных функций – указатель на буфер вывода (например, *pBuf*) и указатель на адрес участка памяти, который будет содержать длину произведенных выходных данных (например, *pulBufLen*). Приложение может вызвать такую функцию двумя способами:

1. Если в *pBuf* содержится значение NULL_PTR, то все, что делает функция, – это выдает (в **pulBufLen*) определенное количество байт, которых будет достаточно для хранения выходных криптографических данных, произведенных из массива, поданного на вход функции. Это количество байт может несколько превышать количество необходимых байт, однако превышение не должно быть большим. Функция выдает значение CKR_OK.

2. Если в *pBuf* содержится не NULL_PTR, то в **pulBufLen* должна содержаться длина в байтах буфера, на который указывает *pBuf*. Если размер этого буфера достаточен для хранения в нем выходных криптографических данных, произведенных из массива, поданного на вход функции, то произведенные данные будут размещены в буфере, а функция выдаст значение CKR_OK. Если размер буфера недостаточен, будет выдано значение CKR_BUFFER_TOO_SMALL. В любом случае **pulBufLen* задается для хранения *точного* количества байт, необходимых для размещения выходных криптографических данных, произведенных из массива, поданного на вход функции.

Следующее напрямую относится ко всем функциям, которые соответствуют описанным выше нормам.

Криптографические функции, возвращаемые данные которых находятся в буфере переменной длины, должны всегда выдавать столько данных, сколько может быть вычислено к определенному моменту из массива, поданного на вход функции. Например, рассмотрим сеанс, который выполняет операцию многостороннего расшифрования по стандарту DES в режиме цепочки шифрблоков с заполнением PKCS. Допустим, вначале 8 байт шифртекста передаются функции **C_DecryptUpdate**. Размер блока DES – 8 байт, однако, ввиду применения заполнения PKCS, на данном этапе неясно, производился шифртекст шифрованием 0-байтовой строки либо шифрованием некоторой строки длиной минимум 8 байт. Следовательно, вызов **C_DecryptUpdate** должен привести к выдаче 0 байт открытого текста. Если со следующим вызовом **C_DecryptUpdate** получит один дополнительный байт шифртекста, результатом вызова станет выдача 8 байт открытого текста (один полный DES-блок).

11.3 Заявление по примерам программного кода

В оставшейся части данного раздела мы перечислим различные функции, описанные в Cryptoki. Большинство функций будет представлено в использовании как минимум в одном примере фрагмента программного кода. В целях краткости примеры программного кода зачастую будут неполными. В частности, в примерах кода в основном будут опускаться возможные возвращаемые значения с кодами ошибок от функций библиотеки C, а также в них не будет возвращаемых значений с кодами ошибок от функций Cryptoki, выдаваемых в реальном режиме.

11.4 Функции общего назначения

Cryptoki предоставляет следующие функции общего назначения:

- **C_Initialize**

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

C_Initialize инициализирует библиотеку Cryptoki. *pInitArgs* имеет значение NULL_PTR или указывает на структуру **CK_C_INITIALIZE_ARGS**, содержащую информацию о том, как библиотека должна действовать с многопоточным доступом. Если приложение не будет получать доступ к Cryptoki через несколько потоков одновременно, оно в общем случае может предоставить функции **C_Initialize** значение NULL_PTR (последствия предоставления такого значения будут разъяснены ниже).

Если значение *pInitArgs* не NULL_PTR, функция **C_Initialize** должна привести его к форме **CK_C_INITIALIZE_ARGS_PTR**, а затем перенаправить результирующий указатель на следующие поля функции **CK_C_INITIALIZE_ARGS:** *CreateMutex*, *DestroyMutex*, *LockMutex*, *UnlockMutex*, *flags* и *pReserved*. Для данной версии Cryptoki получаемое таким образом значение *pReserved* должно быть NULL_PTR; если же оно не NULL_PTR, то функция **C_Initialize** должна выдать значение CKR_ARGUMENTS_BAD.

Если в поле «*флаги*» установлен флаг **CKF_LIBRARY_CANT_CREATE_OS_THREADS**, это указывает на то, что потокам приложения, осуществляющим вызовы библиотеки Cryptoki, не разрешается использовать собственные вызовы операционной системы для порождения новых потоков. Иными словами, программный код библиотеки не может создавать собственные потоки. Если при таком ограничении библиотека не может работать надлежащим образом, функция **C_Initialize** должна выдать значение CKR_NEED_TO_CREATE_THREADS.

В вызове функции **C_Initialize** указывается один из четырех различных способов поддержки многопотокового доступа через значение флага **CKF_OS_LOCKING_OK** к таким полям *флагов* и значений полей указателей функции, как *CreateMutex*, *DestroyMutex*, *LockMutex* и *UnlockMutex*:

1. Если этот флаг *не* выставлен, а поля указателей функции *не* предоставлены (т. е. все они имеют значение NULL_PTR), значит, приложение *не будет* пытаться получить доступ к библиотеке Cryptoki через несколько потоков одновременно.

2. Если этот флаг *выставлен,* а поля указателей функции *не* предоставлены (т. е. все они имеют значение NULL_PTR), значит, приложение *будет* пытаться получить многопотоковый доступ к Cryptoki, а библиотеке нужно использовать элементарные действия местной операционной системы для обеспечения безопасного многопотокового доступа. Если библиотека не может сделать этого, функция **C_Initialize** должна выдать значение CKR_CANT_LOCK.

3. Если этот флаг *не* выставлен, а поля указателей функции *предоставлены* (т. е. все они имеют значение, не равное NULL_PTR), это означает, что приложение *будет* пытаться получить многопотоковый доступ к Cryptoki, а библиотеке нужно использовать предоставленные указатели функции, для управления флагами для обеспечения безопасного многопотокового доступа. Если библиотека не может сделать этого, функция **C_Initialize** должна выдать значение CKR_CANT_LOCK.

4. Если этот флаг *выставлен,* а поля указателей функции *предоставлены* (т. е. все они имеют значение, не равное NULL_PTR), это означает, что приложение *будет* пытаться получить многопотоковый доступ к Cryptoki, а библиотеке нужно использовать либо предоставленные указатели функции для управления флагами, либо элементарные действия местной операционной системы для обеспечения безопасного многопотокового доступа. Если библиотека не может сделать этого, функция **C_Initialize** должна выдать значение CKR_CANT_LOCK.

Если некоторые, но не все, из предоставленных указателей функции на **C_Initialize** имеют значение, не равное NULL_PTR, то **C_Initialize** должна выдать значение CKR_ARGUMENTS_BAD.

Вызов **C_Initialize** при *pInitArgs,* выставленном на NULL_PTR, рассматривается как вызов **C_Initialize** при *pInitArgs,* указывающем на **CK_C_INITIALIZE_ARGS**, поля которой *CreateMutex*, *DestroyMutex*, *LockMutex*, *UnlockMutex* и *pReserved* выставлены на NULL_PTR, а поля их *флагов* выставлены на 0.

Вызов **C_Initialize** должен быть первым вызовом Cryptoki, который делает приложение, за исключением вызовов функции **C_GetFunctionList**. То, что фактически делает эта функция, зависит от программной реализации; обычно она может добиться, чтобы Cryptoki инициализировал свои внутренние буферы памяти или любые другие ресурсы, которые ему требуются.

Если несколько приложений используют Cryptoki, каждое из них должно вызвать **C_Initialize**. За каждым вызовом **C_Initialize** должен (в итоге) следовать одиночный вызов **C_Finalize**. Для получения более подробной информации см. 6.6.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_CANT_LOCK, CKR_CRYPTOKI_ALREADY_ INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_NEED_TO_ CREATE_THREADS, CKR_OK.

Пример: см. C_GetInfo.

- **C_Finalize**

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

C_Finalize вызывается для указания на то, что работа приложения с библиотекой Cryptoki завершена. Это должен быть последний вызов Cryptoki приложением. Параметр *pReserved* зарезервирован для будущих версий; в данной версии его необходимо выставить на NULL_PTR (если **C_Finalize**
вызывается со значением *pReserved*, не равным NULL_PTR, она должна будет выдать значение CKR_ARGUMENTS_BAD).

Если несколько приложений используют Cryptoki, каждое из них должно вызвать **C_Finalize**. Каждому вызову приложением функции **C_Finalize** должен предшествовать отдельный вызов **C_Initialize**; между этими двумя вызовами приложение может делать вызовы других функций Cryptoki. Для получения более подробной информации см. 6.6.

*Несмотря на тот факт, что параметры, предоставляемые функции **C_Initialize**, в общем, могут обеспечивать безопасный многопоточный доступ к библиотеке Cryptoki, поведение **C_Finalize** все равно является неопределенным, если эта функция вызывается приложением, в то время как другие потоки приложения выполняют вызовы Cryptoki. Исключение в этом поведении функции **C_Finalize** случается, когда один поток вызывает **C_Finalize**, а другой поток приложения в это время блокирует функцию Cryptoki **C_WaitForSlotEvent**. Когда такое происходит, блокируемый поток становится разблокированным и выдает значение CKR_CRYPTOKI_NOT_INITIALIZED. Для получения более подробной информации см. **C_WaitForSlotEvent**.*

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.

Пример: см. C_GetInfo.

- **C_GetInfo**

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

C_GetInfo выдает общую информацию о Cryptoki. *pInfo* указывает на адрес участка памяти, который получает информацию.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.

Пример:

```
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);
```

♦ **C_GetFunctionList**

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

Функция **C_GetFunctionList** получает указатель на список указателей функций библиотеки Cryptoki. *ppFunctionList* указывает на значение, которым будет получен указатель на структуру **CK_FUNCTION_LIST** библиотеки, которая, в свою очередь, содержит указатели функций для всех подпрограмм Cryptoki API в библиотеке. *Таким образом, полученный указатель может указывать на память, которая принадлежит библиотеке Cryptoki и которая может являться, а может и не являться перезаписываемой.* Так это или нет, не следует совершать попыток произвести запись в данную память.

C_GetFunctionList – единственная функция Cryptoki, которую может вызывать приложение до вызова **C_Initialize**. Она предназначена для облегчения и ускорения совместного использования приложениями библиотек Cryptoki, а также для использования более чем одной библиотеки Cryptoki одновременно.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.

Пример:

```
CK_FUNCTION_LIST_PTR pFunctionList;
CK_C_Initialize pC_Initialize;
CK_RV rv;

/* It's OK to call C_GetFunctionList before calling C_Initialize */
rv = C_GetFunctionList(&pFunctionList);
assert(rv == CKR_OK);
pC_Initialize = pFunctionList -> C_Initialize;

/* Call the C_Initialize function in the library */
rv = (*pC_Initialize)(NULL_PTR);
```

11.5 Функции управления слотом и носителем

Cryptoki предоставляет следующие функции для управления слотом и носителем.

♦ **C_GetSlotList**

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

C_GetSlotList используется для получения списка слотов в системе. *tokenPresent* указывает, включает ли полученный список только те слоты, в которых присутствует носитель (CK_TRUE), или все слоты (CK_FALSE); *pulCount* указывает на адрес в памяти, который получает информацию о количестве слотов.

Приложение может вызвать функцию **C_GetSlotList** двумя способами:

1. Если *pSlotList* выставлен на NULL_PTR, то все, что делает функция **C_GetSlotList**, – это выдает (в **pulCount*) количество слотов, не предоставляя при этом сам список слотов. Содержимое буфера, на который указывает *pulCount* при входе в **C_GetSlotList**, в данном случае не имеет значения, а вызов возвращает значение CKR_OK.

2. Если значение *pSlotList* не NULL_PTR, то поле **pulCount* должно содержать количество байт (исходя из элементов **CK_SLOT_ID**) буфера, на который указывает *pSlotList*. Если размеры этого буфера достаточны, чтобы хранить список слотов, то данный список выдается в него, а также выдается значение CKR_OK. Если размеры недостаточны, вызов **C_GetSlotList** выдаст значение CKR_BUFFER_TOO_SMALL. В любом случае значение **pulCount* выставлено на хранение определенного количества слотов.

В связи с тем, что функция **C_GetSlotList** не назначает никакого собственного места, приложение часто вызывает **C_GetSlotList** дважды (или иногда даже большее количество раз – если приложение пытается получить список всех слотов с носителями, то количество таких слотов может (к сожалению) измениться с момента, когда приложение запрашивает количество таких слотов, до момента, когда приложение запрашивает информацию о самих слотах). Однако многочисленные вызовы **C_GetSlotList** вовсе *не требуются*.

Все слоты, о которых предоставляет информацию **C_GetSlotList**, должны обладать возможностью реагировать на запросы функции **C_GetSlotInfo**, как на запросы к правомочным слотам. Более того, набор слотов, доступных через библиотеку Cryptoki, проверяется во время направления запроса к функции **C_GetSlotList** (аргумент pSlotList – NULL) на прогноз длины списка. Если приложение вызывает **C_GetSlotList** с аргументом pSlotList, не равным NULL, а *затем* пользователь вводит или удаляет аппаратное устройство, измененный список слотов станет явным и действенным лишь после следующего вызова функции **C_GetSlotList** через NULL. Даже если функция **C_ GetSlotList** успешно вызвана таким способом, измененный список слотов в зависимости от реализации библиотеки может быть, а может не быть успешно распознан. В случаях использования некоторых платформ или более ранних библиотек, совместимых с PKCS11, может понадобиться результативный вызов **C_Initialize** или перезапуск всей системы.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK.

Пример:

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

/* Get list of all slots */
rv = C_GetSlotList(CK_FALSE, NULL_PTR, &ulSlotCount);
if (rv == CKR_OK) {
 pSlotList =
 (CK_SLOT_ID_PTR) malloc(ulSlotCount*sizeof(CK_SLOT_ID));
 rv = C_GetSlotList(CK_FALSE, pSlotList, &ulSlotCount);
 if (rv == CKR_OK) {
 /* Now use that list of all slots */
 .
 .
 }

 free(pSlotList);
}

/* Get list of all slots with a token present */
pSlotWithTokenList = (CK_SLOT_ID_PTR) malloc(0);
ulSlotWithTokenCount = 0;
while (1) {
 rv = C_GetSlotList(
 CK_TRUE, pSlotWithTokenList, ulSlotWithTokenCount);
 if (rv != CKR_BUFFER_TOO_SMALL)
 break;
 pSlotWithTokenList = realloc(
```

```
 pSlotWithTokenList,
 ulSlotWithTokenList*sizeof(CK_SLOT_ID));
}

if (rv == CKR_OK) {
 /* Now use that list of all slots with a token present */
 .
 .
}

free(pSlotWithTokenList);
```

♦ **C_GetSlotInfo**

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

Функция **C_GetSlotInfo** получает информацию об определенном слоте системы. *slotID* – ID слота; *pInfo* указывает на адрес участка памяти, получающего информацию о слоте.

Возвращаемые значения: 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.

Пример: см. C_GetTokenInfo.

♦ **C_GetTokenInfo**

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

Функция **C_GetSlotInfo** получает информацию об определенном носителе системы. *slotID* – ID слота носителя; *pInfo* указывает на адрес участка памяти, получающего информацию о носителе.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_ MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_ MEMORY, CKR_OK, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_ RECOGNIZED, CKR_ARGUMENTS_BAD.

Пример:

```
CK_ULONG ulCount;
CK_SLOT_ID_PTR pSlotList;
CK_SLOT_INFO slotInfo;
CK_TOKEN_INFO tokenInfo;
CK_RV rv;

rv = C_GetSlotList(CK_FALSE, NULL_PTR, &ulCount);
if ((rv == CKR_OK) && (ulCount > 0)) {
 pSlotList = (CK_SLOT_ID_PTR) malloc(ulCount*sizeof(CK_SLOT_ID));
 rv = C_GetSlotList(CK_FALSE, pSlotList, &ulCount);
 assert(rv == CKR_OK);

 /* Get slot information for first slot */
 rv = C_GetSlotInfo(pSlotList[0], &slotInfo);
 assert(rv == CKR_OK);

 /* Get token information for first slot */
 rv = C_GetTokenInfo(pSlotList[0], &tokenInfo);
 if (rv == CKR_TOKEN_NOT_PRESENT) {
 .
 .
 }
```

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

♦ **C_WaitForSlotEvent**

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

C_WaitForSlotEvent ожидает события слота, такого как ввод или удаление носителя. Аргумент *flags* задает наличие или отсутствие блокирования вызова функции **C_WaitForSlotEvent** (т. е. ожидает, пока произойдет событие слота); аргумент *pSlot* указывает на адрес участка памяти, куда поступит ID слота, в котором произошло событие. Аргумент *pReserved* зарезервирован для последующих версий; в данной версии Cryptoki его значение должно быть NULL_PTR.

В настоящее время единственным флагом, определенным для использования в аргументе *flags*, является **CKF_DONT_BLOCK**.

Внутри каждое приложение Cryptoki для каждого слота имеет по флагу, используемому, чтобы следить, происходило или нет какое-либо нераспознанное событие, связанное с данным слотом. Когда приложение производит первоначальный вызов **C_Initialize**, каждый слотовый флаг события очищается. Когда бы ни произошло событие слота, устанавливается флаг, соответствующий слоту, в котором происходит событие.

Если вызывается **C_WaitForSlotEvent** с флагом **CKF_DONT_BLOCK**, установленным в аргументе *flags*, и при этом выставлен флаг какого-либо события слота, этот слотовый флаг события очищается, а вызов выдается вместе с ID данного слота в область памяти, на адрес которой указывает аргумент *pSlot.* Если более чем один флаг события слота выставлен в момент вызова, один из таких слотов выбирается библиотекой, чтобы очистить его флаг события и получить возвращаемый ID этого слота.

Если **C_WaitForSlotEvent** вызывается, когда флаг **CKF_DONT_BLOCK** выставлен в аргументе *flags*, а ни один флаг события слота не выставлен, то вызов возвращается со значением CKR_NO_EVENT. В данном случае содержимое ячейки памяти указывало на *pSlot*, когда **C_WaitForSlotEvent** был неопределен.

Если **C_WaitForSlotEvent** вызывается с очищенным флагом **CKF_DONT_BLOCK** в аргументе *flags*, то вызов поведет себя, как описано выше, за исключением того, что он будет блокироваться. Это означает, что, если в момент вызова флаг события слота выставлен не был, **C_WaitForSlotEvent** будет ожидать, пока какой-либо флаг события слота не будет выставлен. Если поток приложения выполняет вызов **C_WaitForSlotEvent**, который блокируется, когда другой поток этого приложения вызывает **C_Finalize**, вызов **C_WaitForSlotEvent** возвращает значение CKR_CRYPTOKI_NOT_INITIALIZED.

*Хотя параметры, предоставленные функции **C_Initialize**, могут в общем случае обеспечить безопасный многопоточный доступ к библиотеке Cryptoki, исключительность **C_WaitForSlotEvent** состоит в том, что поведение Cryptoki является неопределенным, если несколько потоков одного приложения осуществляют одновременный вызов **C_WaitForSlotEvent**.*

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_NO_EVENT, CKR_OK.

Пример:

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

.
.
/* Block and wait for a slot event */
rv = C_WaitForSlotEvent(flags, &slotID, NULL_PTR);
assert(rv == CKR_OK);

/* See what's up with that slot */
rv = C_GetSlotInfo(slotID, &slotInfo);
```

```
assert(rv == CKR_OK);
.
.
```

♦ **C_GetMechanismList**

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

C_GetMechanismList используется для получения списка типов механизмов, поддерживаемых носителем. *SlotID* – ID слота носителя; *pulCount* указывает на ячейку памяти, которая получает информацию о количестве механизмов.

Приложение может вызвать функцию **C_GetMechanismList** двумя способами:

1. Если *pMechanismList* выставлен на NULL_PTR, то все, что делает функция **C_GetMechanism-List,** – это выдает (в **pulCount*) количество механизмов, не предоставляя при этом самого списка слотов. Содержимое **pulCount* при входе в **C_GetMechanismList** в данном случае значения не имеет, а вызов выдает значение CKR_OK.
2. Если значение *pMechanismList* не NULL_PTR, то поле **pulCount* должно содержать количество байтов (исходя из элементов **CK_MECHANISM_TYPE**) буфера, на который указывает *pMechanismList*. Если размеры этого буфера достаточны, чтобы хранить список механизмов, то данный список выдается в него, а также выдается значение CKR_OK. Если размеры недостаточны, вызов **C_Get-MechanismList** выдаст значение CKR_BUFFER_TOO_SMALL. В любом случае значение **pulCount* выставлено на сохранение определенного количества механизмов.

Так как **C_GetMechanismList** не назначает никакого собственного места в памяти, приложение зачастую будет вызывать **C_GetMechanismList** дважды. Однако такое поведение вовсе не обязательно.

Выдаваемые значения: CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_ DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_ GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_ PRESENT, CKR_TOKEN_NOT_RECOGNIZED, CKR_ARGUMENTS_BAD.

Пример:

```
CK_SLOT_ID slotID;
CK_ULONG ulCount;
CK_MECHANISM_TYPE_PTR pMechanismList;
CK_RV rv;

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

♦ **C_GetMechanismInfo**

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

C_GetMechanismInfo получает информацию об определенном механизме, предположительно поддерживаемом носителем. *slotID* – ID слота носителя; *type* – тип механизма; *pInfo* указывает на ячейку в памяти, которая получает информацию о механизме.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MECHANISM_INVALID, CKR_OK, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_RECOGNIZED, CKR_ARGUMENTS_BAD.

Пример:

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

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

- **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 инициализирует носитель. *slotID* – ID слота носителя; *pPin* указывает на начальный PIN SO (*не должен* заканчиваться нулем); *ulPinLen* – длина PIN в байтах; *pLabel* указывает на 32-битную метку носителя (которая дополняется пустыми символами и которая *не должна* заканчиваться нулем). Настоящий стандарт позволяет значениям PIN содержать любой разрешенный символ UTF8, однако ограничения по подмножеству могут задаваться носителем.

Если носитель не был инициализирован (т. е. он новый, в заводском состоянии), параметр *pPin* становится начальным значением PIN SO. Если носитель проходит повторную инициализацию, параметр *pPin* сверяется с существующим PIN SO для получения разрешения на операцию инициализации. В обоих случаях PIN SO имеет значение *pPin* после того, как функция успешно завершила работу. Если PIN SO утерян, карту следует инициализировать повторно с использованием механизма, который не рассматривается в настоящем стандарте. Флаг **CKF_TOKEN_INITIALIZED** в структуре **CK_TOKEN_INFO** указывает на действие, которое будет выполнено в результате вызова функции **C_InitToken**. Если он выставлен, носитель будет повторно инициализирован, а клиент должен при этом ввести существующий пароль SO в *pPin*.

Когда носитель инициализируется, все объекты, которые можно уничтожить, уничтожаются (т. е. все, за исключением «неразрушаемых» объектов, таких как ключи, встроенные в носитель). Кроме того, доступ обычного пользователя становится невозможным до тех пор, пока SO не введет PIN этого пользователя. В зависимости от носителя могут создаваться некоторые объекты «по умолчанию», а атрибуты некоторых объектов могут выставляться на значения по умолчанию.

Если носитель имеет «защищенный путь аутентификации», на что указывает флаг **CKF_PROTECTED_AUTHENTICATION_PATH**, выставленный в **CK_TOKEN_INFO**, это означает, что пользователь имеет возможность пройти аутентификацию в носителе без необходимости отправления приложением его PIN через библиотеку Cryptoki. Одна из таких возможностей состоит в том, что пользователь может ввести PIN через интерфейс PINpad на самом носителе или на слоте устройства. Чтобы инициализировать носитель с использованием такого защищенного пути, значение параметра *pPin* к **C_InitToken** должно быть NULL_PTR. Во время выполнения **C_InitToken** PIN SO будет вводиться через защищенный путь аутентификации.

Если в носителе есть защищенный путь аутентификации, отличный от PINpad, то ответ на вопрос о возможности использования **C_InitToken** для инициализации носителя зависит от самого носителя.

Невозможно проинициализировать носитель, если Cryptoki выявит, что у *любого* приложения есть открытый сеанс с ним; когда вызов функции **C_InitToken** осуществляется в таких обстоятельствах, этот вызов будет неудачным и закончится выдачей кода ошибки CKR_SESSION_EXISTS. К сожалению, при вызове **C_InitToken** может произойти так, что какое-либо другое приложение *действительно* имеет открытый сеанс с носителем, но Cryptoki не может выявить этого, так как он не может установить ничего относительно других приложений, использующих носитель. В этом случае последствия вызова **C_InitToken** не определены.

Функции **C_InitToken** может не быть достаточно для того, чтобы правильно инициализировать сложные носители. При таких ситуациях должен применяться механизм инициализации, который находится вне области действия Cryptoki. Описание «сложного носителя» является особенностью продукта.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_ MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_ GENERAL_ ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_PIN_INCORRECT, CKR_PIN_LOCKED, CKR_ SESSION_EXISTS, CKR_SLOT_ID_INVALID, CKR_TOKEN_NOT_PRESENT, CKR_TOKEN_NOT_ RECOGNIZED, CKR_TOKEN_WRITE_PROTECTED, CKR_ARGUMENTS_BAD.

Пример:

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

.
.
memset(label, ‘ ’, sizeof(label));
memcpy(label, “My first token”, strlen(“My first token”));
rv = C_InitToken(slotID, pin, strlen(pin), label);
if (rv == CKR_OK) {
 .
 .
}
```

- **C_InitPIN**

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

Функция **C_InitPIN** инициализирует PIN обычного пользователя. *hSession* – идентификатор сеанса; *pPin* указывает на PIN обычного пользователя; *ulPinLen* – длина PIN в байтах. Настоящий стандарт позволяет значениям PIN содержать любой разрешенный символ UTF8, однако ограничения по подмножеству могут задаваться носителем.

C_InitPIN может вызываться только в состоянии «Сеанс функций SO чтения/записи». Попытка вызвать ее из сеанса в любом ином состоянии завершится неудачно с кодом ошибки CKR_USER_ NOT_LOGGED_IN.

Если носитель имеет «защищенный путь аутентификации», на что указывает флаг **CKF_PROTECTED_AUTHENTICATION_PATH**, выставленный в **CK_TOKEN_INFO**, это означает, что пользователь имеет возможность пройти аутентификацию в носителе без необходимости отправления приложением его PIN через библиотеку Cryptoki. Одна из таких возможностей состоит в том, что пользователь может ввести PIN через интерфейс PINpad на самом носителе или на слоте устройства. Чтобы инициализировать PIN обычного пользователя на носителе с использованием такого защищенного пути, значение параметра *pPin* к **C_InitPIN** должно быть NULL_PTR. Во время выполнения **C_InitPIN** PIN SO будет вводиться через защищенный путь аутентификации.

Если в носителе есть защищенный путь аутентификации, отличный от PINpad, то ответ на вопрос о возможности использования **C_InitPIN** для инициализации носителя зависит от самого носителя.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_ DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_PIN_INVALID, CKR_PIN_LEN_RANGE, CKR_SESSION_CLOSED, CKR_SESSION_READ_ONLY, CKR_SESSION_HANDLE_INVALID, CKR_TOKEN_ WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN, CKR_ARGUMENTS_BAD.

Пример:

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

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

♦ **C_SetPIN**

```
CK_DEFINE_FUNCTION(CK_RV, C_SetPIN)(
  CK_SESSION_HANDLE hSession,
  CK_UTF8CHAR_PTR pOldPin,
  CK_ULONG ulOldLen,
  CK_UTF8CHAR_PTR pNewPin,
  CK_ULONG ulNewLen
);
```

Функция **C_SetPIN** изменяет PIN пользователя, который зарегистрирован на текущий момент, или используется CKU_USER PIN, если сеанс не зарегистрирован. *hSession* – идентификатор сеанса; *pOldPin* указывает на старый PIN; *ulOldLen* – длина в байтах старого PIN; *pNewPin* указывает на новый PIN; *ulNewLen* – это длина нового PIN в байтах. Настоящий стандарт позволяет значениям PIN содержать любой разрешенный символ UTF8, однако ограничения по подмножеству могут задаваться носителем.

C_SetPIN может вызываться только в состоянии «Открытый сеанс чтения/записи» или в состоянии «Сеанс пользовательских функций чтения/записи». Попытка вызвать ее из сеанса в любом ином состоянии завершится неудачно с кодом ошибки CKR_USER_NOT_LOGGED_IN.

Если носитель имеет «защищенный путь аутентификации», на что указывает флаг **CKF_PROTECTED_AUTHENTICATION_PATH**, выставленный в **CK_TOKEN_INFO**, это означает, что пользователь имеет возможность пройти аутентификацию в носителе без необходимости отправления приложением его PIN через библиотеку Cryptoki. Одна из таких возможностей состоит в том, что пользователь может ввести PIN через интерфейс PINpad на самом носителе или на слоте устройства. Чтобы изменить PIN обычного пользователя на носителе с использованием такого защищенного пути, значение параметра *pOldPin* и *pNewPin* к **C_InitPIN** должно быть NULL_PTR. Во время выполнения **C_SetPIN** текущий пользователь введет старый PIN и новый PIN через защищенный путь аутентификации. Не указывается, как для ввода *двух* кодов должен использоваться PINpad; это может происходить по-разному.

Если в носителе есть защищенный путь аутентификации, отличный от PINpad, то ответ на вопрос о возможности использования **C_SetPIN** для инициализации носителя зависит от самого носителя.

Возвращаемые значения: 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.

Пример:

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

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

11.6 Функции управления сеансом

Типовое приложение может выполнять следующую последовательность действий для использования носителя (следует отметить, что есть и другие разумные последовательности действий, которые приложение могло бы выполнять):

1. Выбрать носитель.
2. Сделать один или более вызовов функции **C_OpenSession** для установления одного или более сеанса с носителем.
3. Вызвать **C_Login** для регистрации пользователя в носителе. Так как все сеансы, которые приложение ведет с носителем, имеют общее состояние регистрации, функция **C_Login** должна вызываться только для одного сеанса.
4. Выполнить криптографические операции, используя сеансы с носителем.
5. Вызвать **C_CloseSession** один раз для каждого сеанса, имеющегося у приложения с носителем, или вызвать **C_CloseAllSessions**, чтобы закрыть все сеансы приложения одновременно.

Как уже говорилось ранее, приложение может иметь одновременные сеансы с более чем одним носителем. Возможно также, что носитель будет иметь одновременные сеансы с более чем одним приложением.

Cryptoki предоставляет следующие функции для управления сеансом:

- **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 открывает сеанс между приложением и носителем в определенном слоте.

slotID – ID слота; *flags* указывает на тип сеанса; *pApplication* – заданный приложением указатель, который должен передаваться в ответ на обратный вызов уведомления; *Notify* – адрес функции обратного вызова уведомления (см. 11.17); *phSession* указывает на адрес участка памяти, куда поступает идентификатор для нового сеанса.

При открытии сеанса с использованием функции **C_OpenSession** параметр *flags* состоит из логического ИЛИ (OR) либо из битовых флагов в количестве от нуля и более, заданных в типе данных **CK_SESSION_INFO**. Исходя из соображений наследования, бит **CKF_SERIAL_SESSION** должен всегда быть выставлен; если в вызове функции **C_OpenSession** такой бит не выставлен, вызов должен возвращаться невыполненным с кодом ошибки CKR_PARALLEL_NOT_SUPPORTED.

На количество одновременных сеансов, которые приложение может иметь с одним носителем, может налагаться ограничение в зависимости от того, является ли сеанс сеансом только для чтения или же это сеанс чтения/записи. При попытке открыть сеанс, которая не увенчалась успехом в связи с тем, что существует слишком много открытых сеансов определенного типа, должно возвращаться значение CKR_SESSION_COUNT.

Если носитель защищен от записи (как задано в структуре **CK_TOKEN_INFO**), то с этим носителем могут открываться лишь сеансы «только для чтения».

Если приложение, вызывающее **C_OpenSession**, уже открыло сеанс SO чтения/записи с носителем, то любая попытка открыть с этим носителем сеанс только для чтения завершится неудачно, с кодом ошибки CKR_SESSION_READ_WRITE_SO_EXISTS (см. 6.7.7).

Обратный вызов *Notify* используется Cryptoki для уведомления приложения о каких-либо событиях. Если приложение не желает поддерживать обратные вызовы, оно должно передать значение NULL_PTR в качестве параметра *Notify*. Для получения более подробной информации см. 11.17.

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

Пример: см. C_CloseSession.

- **C_CloseSession**

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

Функция **C_CloseSession** закрывает сеанс между приложением и носителем. *hSession* – идентификатор сеанса.

Когда сеанс закрывается, все объекты, созданные данным сеансом, уничтожаются автоматически, даже если приложение имеет другие сеансы, «использующие» данные объекты (для получения более подробной информации см. 6.7.5 – 6.7.7).

Если данная функция является успешной и закрывает последний сеанс между приложением и носителем, регистрационное состояние носителя по отношению к данному приложению возвращается в «открытые сеансы». Любые новые сеансы для носителя, которые откроют приложение, будут открытыми сеансами только для чтения (R/O Public) либо открытыми сеансами чтения/записи (R/W Public).

В зависимости от носителя после закрытия последнего открытого сеанса, имевшегося с этим носителем у любого приложения, носитель может быть «извлечен» из своего считывающего устройства (если такая возможность существует).

Несмотря на то, что данную функцию **C_CloseSession** предполагается использовать для закрытия сеансов, возвращаемое значение CKR_SESSION_CLOSED является *кодом ошибки*. Фактически оно (как более или менее вероятный вариант) уведомляет о следующем событии: во время выполнения запроса данной функции был сделан еще один запрос функции **C_CloseSession** (на закрытие данного конкретного сеанса), и этот запрос в приоритетном порядке был удовлетворен. Такое использование сеансов – «плохая идея», и Cryptoki не берется прогнозировать общие последствия действий такого рода со стороны приложения.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример:

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

.
.
application = 17;
MyNotify = &EncryptionSessionCallback;
rv = C_OpenSession(
 slotID, CKF_SERIAL_SESSION | CKF_RW_SESSION,
         (CK_VOID_PTR) &application, MyNotify,
 &hSession);
if (rv == CKR_OK) {
 .
 .
 C_CloseSession(hSession);
}
```

- **C_CloseAllSessions**

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

C_CloseAllSessions закрывает все сеансы, имеющиеся у приложения с носителем. *slotID* задает

слот носителя.

Когда сеанс закрывается, все объекты, созданные данным сеансом, уничтожаются автоматически.

После успешного выполнения данной функции регистрационное состояние носителя по отношению к данному приложению возвращается в «открытые сеансы». Любые новые сеансы для носителя, которые откроет приложение, будут открытыми сеансами только для чтения (R/O Public) либо открытыми сеансами чтения/записи (R/W Public).

В зависимости от носителя после закрытия последнего открытого сеанса, имевшегося с этим носителем у любого приложения, носитель может быть «извлечен» из своего считывающего устройства (если такая возможность существует).

Возвращаемые значения: 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.

Пример:

```
CK_SLOT_ID slotID;
CK_RV rv;

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

♦ **C_GetSessionInfo**

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

Функция **C_GetSessionInfo** получает информацию о сеансе. *hSession* – идентификатор сеанса; *pInfo* указывает на адрес участка памяти, который получает информацию о сеансе.

Возвращаемые значения: 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.

Пример:

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

♦ **C_GetOperationState**

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

Функция **C_GetOperationState** получает копию состояния криптографических операций сеанса, закодированную как байтовая строка. *hSession* – идентификатор сеанса; *pOperationState* указывает на

участок памяти, куда поступает информация о состоянии; *pulOperationStateLen* указывает на участок памяти, куда поступает информация о длине состояния в байтах.

Хотя сохраненные функцией **C_GetOperationState** выходные данные состояния реально «криптографическим механизмом» не производятся, **C_GetOperationState** при осуществлении вывода данных тем не менее руководствуется порядком, описанным в 11.2.

Конкретное «состояние криптографических операций», сохраняемое данной функцией, варьируется от носителя к носителю; однако данное состояние предоставляется в качестве входных данных функции **C_SetOperationState** для восстановления криптографических действий сеанса.

Рассмотрим сеанс, который выполняет операцию вычисления списка, используя SHA-1 (т. е. сеанс использует механизм **CKM_SHA_1**). Предположим, что операция вычисления списка была инициализирована надлежащим образом и что точно 80 байт данных предоставлено к определенному моменту в качестве данных ввода для SHA-1. Сейчас приложение желает «сохранить состояние» данной операции вычисления списка, с тем чтобы оно могло ее продолжить позже. В данном конкретном случае ввиду того, что SHA-1 обрабатывает за один прием 512 бит (64 байта) данных ввода, состояние криптографической операции сеанса, скорее всего, будет состоять из трех раздельных частей: состояния 160-битовой внутренней сцепленной переменной SHA-1; 16 байтов необработанных данных ввода и некоторых административных данных, указывающих на то, что это сохраненное состояние исходит от сеанса, который выполнял операцию вычисления списка SHA-1. Трех этих вместе взятых порций информации достаточно для того, чтобы текущая операция вычисления списка была продолжена позже.

Рассмотрим следующий сеанс, выполняющий операцию шифрования DES (блочный шифр с длиной блока в 64 бита) в режиме цепочки шифрблоков (Cipher-Block Chaining, CBC) (т. е. сеанс использует механизм **CKM_DES_CBC**). Допустим, что ровно 22 байта данных ввода (кроме вектора инициали-зации (IV) для режима CBC) были переданы к определенному моменту времени как данные ввода для DES, а это означает, что первые два 8-байтных блока шифртекста уже сформированы в качестве данных вывода. В данном случае состояние криптографических операций сеанса, скорее всего, состоит из трех или четырех раздельных частей: второго 8-байтного блока шифртекста (он будет использоваться в режиме CBC для создания следующего блока шифртекста); 6 байт данных, еще ожидающих шифрования; некоторых административных данных, указывающих на то, что это сохраненное состояние исходит из сеанса, который выполнял DES-шифрование в режиме CBC, и, возможно, DES-ключа, используемого для шифрования (для получения более подробной информации о присутствии или отсутствии ключа в сохраненном сеансе см. **C_SetOperationState**).

Если сеанс выполняет две криптографические операции одновременно (см. 11.13), состояние криптографических операций сеанса будет содержать всю необходимую информацию для восстановления обеих операций.

Попытка сохранить состояние криптографических операций сеанса, который в данный момент не содержит никаких активных потенциально сохраняемых криптографических операций (шифрование, расшифрование, получение списка, постановка электронной подписи без восстановления сообщения, верификация без восстановления сообщения или любая другая допустимая комбинация двух операций из перечисленных выше), завершится неудачно с кодом ошибки CKR_OPERATION_NOT_INITIALIZED.

Попытка сохранить состояние криптографических операций сеанса, который выполняет соответствующую криптографическую операцию (или две таких операции), но не может их выполнить в силу одной из различных причин (например, необходимая информация о сеансе и/или о ключе не может покидать пределы носителя), завершится сбоем с кодом ошибки CKR_STATE_UNSAVEABLE.

Возвращаемые значения: CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_ DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_ GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_ SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_STATE_UNSAVEABLE, CKR_ ARGUMENTS_BAD.

Пример: см. C_SetOperationState.

- **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 восстанавливает состояние криптографических операций сеанса из байтовой строки, полученной при помощи **C_GetOperationState**. *hSession* – идентификатор сеанса. *pOperationState* указывает на участок памяти, в котором хранится информация о сохраненном состоянии; *ulOperation-StateLen* содержит длину сохраненного состояния; *hEncryptionKey* содержит идентификатор ключа, который будет использоваться в текущей операции шифрования или расшифрования в восстановленном сеансе (либо 0, если ключ шифрования или расшифрования не нужен, если такая операция не выполняется в сохраненном сеансе или если вся необходимая информация о ключе присутствует в сохраненном состоянии); *hAuthenticationKey* содержит идентификатор ключа, который будет использоваться для текущей операции постановки электронной подписи, MAC или верификации в восстановленном сеансе (или 0, если такой ключ не нужен, в связи с тем, что такие операции не выполняются в сохраненном сеансе, или если вся необходимая информация о ключе присутствует в сохраненном состоянии).

Состояние не нужно получать из того же сеанса (далее – сеанс-источник), в который оно будет восстанавливаться (далее – сеанс-назначение). Однако сеанс-источник и сеанс-назначение должны быть в общем сеансовом состоянии (*например,* CKS_RW_USER_FUNCTIONS) и должны работать с общим носителем. Также нет гарантии, что состояние криптографических операций сможет переноситься между различными регистрациями или между различными реализациями Cryptoki.

Если функция **C_SetOperationState** получает якобы сохраненное состояние криптографических операций, которое она определяет как недействительное сохраненное состояние (или как состояние криптографических операций из сеанса с иным состоянием, или как состояние криптографических операций от другого носителя), то эта функция завершается сбоем с кодом ошибки CKR_SAVED_ STATE_INVALID.

Сохраненное состояние, получаемое при помощи запросов к **C_GetOperationState**, может содержать, а может не содержать информацию о ключах, используемых для текущих криптографических операций. Если сохраненное состояние криптографических операций имеет текущую операцию шифрования или расшифрования, а используемый для данной операции ключ в этом состоянии не сохранен, он должен быть предоставлен функции **C_SetOperationState** в аргументе *hEncryptionKey.* Если он предоставлен не будет, функция **C_SetOperationState** завершится сбоем с кодом ошибки CKR_KEY_ NEEDED. Если используемый для осуществления операции ключ *сохранен* в состоянии, то он *может* быть предоставлен в аргумент *hEncryptionKey*, но это не обязательно.

Аналогично, если сохраненное состояние криптографических операций содержит текущую операцию постановки электронной подписи, MAC или верификации, а используемый для операции ключ не сохранен, его нужно предоставить функции **C_SetOperationState** в аргументе *hAuthenticationKey.* Если ключ предоставлен не будет, функция **C_SetOperationState** завершится сбоем с кодом ошибки CKR_KEY_ NEEDED. Если используемый для осуществления операции ключ *сохранен*, его *можно* предоставить в аргумент *hAuthenticationKey*, но это не обязательно.

Если в ответ на вызов функции **C_SetOperationState** предоставлен *несоответствующий* ключ (*например,* в аргумент *hEncryptionKey* предоставляется ненулевой идентификатор ключа, но при этом сохраненное состояние криптографических операций не содержит текущих операций шифрования или расшифрования), функция **C_SetOperationState** завершится сбоем с кодом ошибки CKR_KEY_NOT_ NEEDED.

Если ключ предоставляется функции **C_SetOperationState** в качестве аргумента и эта функция может некоторым образом установить, что данный ключ не являлся ключом, который использовался в сеансе-источнике для состояния криптографических операций (она, возможно, и сможет выявить это, если, например, в сохраненном состоянии имеется сам ключ или хэш ключа), то функция **C_SetOperationState** завершится сбоем с кодом ошибки CKR_KEY_CHANGED.

Приложение может поискать носитель во флаге **CKF_RESTORE_KEY_NOT_NEEDED** флагового поля **CK_TOKEN_INFO,** чтобы установить, нуждается ли носитель в предоставлении идентификаторов ключей в ответ на вызовы функции **C_SetOperationState**. *Во всех случаях*, когда данный флаг выставлен на «истинно», вызов **C_SetOperationState** не нуждается в предоставлении ему идентификатора ключа. Если данный флаг выставлен на «ложно», то по крайней мере на некоторое время

функции **C_SetOperationState** нужен идентификатор ключа и поэтому приложение, вероятно, должно *всегда* передавать любые значимые идентификаторы ключей при восстановлении состояния криптографических операций в сеанс.

C_SetOperationState может успешно восстанавливать состояние криптографических операций сеанса, даже если этот сеанс содержит активные криптографические операции или операции поиска объектов при вызове **C_SetOperationState** (текущие операции немедленно останавливаются).

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

Пример:

```
CK_SESSION_HANDLE hSession;
CK_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 */
 .
 .
}
```

♦ **C_Login**

```
CK_DEFINE_FUNCTION(CK_RV, C_Login)(
```

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

C_Login регистрирует пользователя в носителе. *hSession* – идентификатор сеанса; *userType* – тип пользователя; *pPin* указывает на PIN пользователя; *ulPinLen* – длина PIN. Настоящий стандарт позволяет значениям PIN содержать любой разрешенный символ UTF8, однако ограничения по подмножеству могут задаваться носителем.

Когда при вызове тип пользователя – CKU_SO или CKU_USER, каждый из сеансов приложения войдет в одно из следующих состояний: «Сеанс функций SO чтения/записи», «Сеанс пользовательских функций чтения/записи» или «Сеанс пользовательских функций только для чтения». Если тип пользователя – это CKU_CONTEXT_SPECIFIC, то поведение C_Login зависит от контекста, в котором она вызывается. Ненадлежащее применение этого типа пользователя приведет к выдаче значения CKR_OPERATION_NOT_INITIALIZED.

Если носитель имеет «защищенный путь аутентификации», на что указывает флаг **CKF_PROTECTED_AUTHENTICATION_PATH**, выставленный в **CK_TOKEN_INFO**, это означает, что пользователь имеет возможность пройти аутентификацию в носителе без необходимости отправления приложением его PIN через библиотеку Cryptoki. Одна из таких возможностей состоит в том, что пользователь может ввести PIN через интерфейс PINpad на самом носителе или на слоте устройства. Пользователь может и не использовать PIN – аутентификация может быть осуществлена, например, через дактилоскопическое устройство. Чтобы инициализировать носитель с использованием такого защищенного пути, значение параметра *pPin* к **C_InitToken** должно быть NULL_PTR. При возврате функции **C_Login**, какой бы метод аутентификации из поддерживаемых носителем ни использовался, возвращаемое значение CKR_OK означает, что аутентификация пользователя прошла успешно, а возвращаемое значение CKR_PIN_INCORRECT означает, что пользователю отказано в доступе.

Если в сеансе приложения имеются какие-либо активные криптографические функции или функции поиска объектов, а **C_Login** при этом успешно выполнена приложением, то эти операции могут еще быть, а могут уже не быть активными. Следовательно, до регистрации любые активные операции должны быть завершены.

Если приложение, вызывающее функцию **C_Login**, имеет с носителем открытый сеанс только для чтения, оно не сможет зарегистрировать SO в сеансе (см. 6.7.7). Попытка сделать это приведет к сбою с кодом ошибки CKR_SESSION_READ_ONLY_EXISTS.

C_Login может вызываться многократно, без вмешательства в выполнение вызовов функции **C_Logout**, тогда (и только тогда), когда существует ключ с атрибутом CKA_ALWAYS_AUTHENTICATE, выставленным на CK_TRUE, а пользователю нужно выполнить криптографические операции над этим ключом. См. далее 11.17.

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

Пример: см. C_Logout.

♦ **C_Logout**

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

C_Logout выводит пользователя из носителя. *hSession* – идентификатор сеанса.

В зависимости от типа текущего пользователя, если следует вызов, все сеансы приложения войдут в состояние «открытый сеанс чтения/записи» или «открытый сеанс только для чтения».

Когда **C_Logout** выполняется успешно, любой из идентификаторов приложения, указывающих на секретные объекты, становится недействительным (даже если пользователь в дальнейшем снова зарегистрируется в носителе, данные идентификаторы останутся недействительными). Кроме того, все секретные сеансовые объекты из сеансов, принадлежащих приложению, уничтожаются.

Если в сеансе приложения имеются какие-либо активные криптографические операции или операции поиска объектов, а затем приложением успешно выполнена функция **C_Logout**, эти операции могут еще быть, а могут уже не быть активными. Следовательно, до выхода любые активные операции должны быть завершены.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_ DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_ USER_NOT_LOGGED_IN.

Пример:

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

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

11.7 Функции управления объектами

Для управления объектами Cryptoki предоставляет перечисленные ниже функции. Описание дополнительных функций специально для управления объектами ключей дано в 11.14.

- **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 создает новый объект. *hSession* – идентификатор сеанса; *pTemplate* указывает на шаблон объекта; *ulCount* – количество атрибутов в шаблоне; *phObject* указывает на адрес участка памяти, которая получает идентификатор нового объекта.

Если вызов **C_CreateObject** не может поддерживать точного шаблона, предоставляемого ей, данная функция завершится сбоем и на выходе не создаст объекта.

Если **C_CreateObject** используется для создания объекта ключа, то в этом объекте ключа атрибут **CKA_LOCAL** будет выставлен на CK_FALSE. Если данный объект ключа является секретным или личным/секретным ключом, то в новом ключе атрибут **CKA_ALWAYS_SENSITIVE** будет выставлен на CK_FALSE и атрибут **CKA_NEVER_EXTRACTABLE** будет выставлен на это же значение.

В ходе сеанса только для чтения можно создавать только сеансовые объекты. Если обычный пользователь не прошел регистрацию, создаваться могут только открытые объекты.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_DOMAIN_PARAMS_INVALID, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_ TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

Пример:

```
CK_SESSION_HANDLE hSession;
```

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

.
.
/* Create a data object */
rv = C_CreateObject(hSession, &dataTemplate, 4, &hData);
if (rv == CKR_OK) {
  .
  .
}

/* Create a certificate object */
rv = C_CreateObject(
 hSession, &certificateTemplate, 5, &hCertificate);
if (rv == CKR_OK) {
  .
  .
}

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

♦ **C_CopyObject**

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

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

C_CopyObject копирует объекты, создавая в качестве копии новый объект. *hSession* – идентификатор сеанса; *hObject* – идентификатор объекта; *pTemplate* указывает на шаблон для нового объекта; *ulCount* – количество атрибутов в шаблоне; *phNewObject* указывает на адрес участка памяти, в котором сохраняется идентификатор копии объекта.

В шаблоне могут быть определены новые значения для любых атрибутов объекта, которые обычно могут модифицироваться (*например,* в процессе копирования секретного ключа атрибут ключа **CKA_EXTRACTABLE** можно изменить с CK_TRUE на CK_FALSE, но не в ином порядке). Если вносится такое изменение, новый атрибут ключа **CKA_NEVER_EXTRACTABLE** будет иметь значение CK_FALSE. Точно так же в шаблоне может быть указано, что атрибут нового ключа **CKA_SENSITIVE** должен быть выставлен на CK_TRUE; новый ключ будет иметь то же значение своего атрибута **CKA_ALWAYS_ SENSITIVE**, что и исходный ключ. Также в нем могут быть указаны новые значения для атрибутов **CKA_TOKEN** и **CKA_PRIVATE** (*например*, скопировать объект сеанса в объект носителя). Если в шаблоне указано значение атрибута, которое несовместимо с существующими атрибутами объекта, вызов завершится сбоем с кодом ошибки CKR_TEMPLATE_INCONSISTENT.

Если вызов **C_CopyObject** не может поддерживать точного шаблона, предоставляемого ей, данная функция завершится сбоем и на выходе не создаст объекта.

В ходе сеанса «только для чтения» можно создавать только сеансовые объекты. Если обычный пользователь не прошел регистрацию, создаваться могут только открытые объекты.

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

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey, hNewKey;
CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_DES;
CK_BYTE id[] = {...};
CK_BYTE keyValue[] = {...};
CK_BBOOL false = CK_FALSE;
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE keyTemplate[] = {
  {CKA_CLASS, &keyClass, sizeof(keyClass)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &false, sizeof(false)},
  {CKA_ID, id, sizeof(id)},
  {CKA_VALUE, keyValue, sizeof(keyValue)}
};
CK_ATTRIBUTE copyTemplate[] = {
  {CKA_TOKEN, &true, sizeof(true)}
};
CK_RV rv;

.
.
/* Create a DES secret key session object */
rv = C_CreateObject(hSession, &keyTemplate, 5, &hKey);
if (rv == CKR_OK) {
  /* Create a copy which is a token object */
  rv = C_CopyObject(hSession, hKey, &copyTemplate, 1, &hNewKey);
  .
  .
```

```
}
```

♦ **C_DestroyObject**

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

C_DestroyObject уничтожает объект. *hSession* – идентификатор сеанса; а *hObject* – идентификатор объекта.

В ходе сеанса «только для чтения» можно уничтожать только сеансовые объекты. Если обычный пользователь не прошел регистрацию, уничтожаться могут только открытые объекты.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TOKEN_WRITE_PROTECTED.

Пример: см. C_GetObjectSize.

♦ **C_GetObjectSize**

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

C_GetObjectSize получает размер объекта в байтах. *hSession* – идентификатор сеанса; *hObject* – идентификатор объекта; *pulSize* указывает на адрес области памяти, которая получит данные о размере объекта в байтах.

Cryptoki не определяет, каково точное значение понятия «размер объекта». Интуитивно, это некоторое количественное выражение того, сколько памяти носителя занимает объект. Если приложение удаляет, предположим, секретный объект размером S, то можно предположить, что поле *ulFreePrivate-Memory* структуры носителя **CK_TOKEN_INFO** увеличивается примерно на S.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_INFORMATION_SENSITIVE, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hObject;
CK_OBJECT_CLASS dataClass = CKO_DATA;
CK_CHAR application[] = {"My Application"};
CK_BYTE dataValue[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &dataClass, sizeof(dataClass)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_APPLICATION, application, sizeof(application)},
  {CKA_VALUE, value, sizeof(value)}
};
CK_ULONG ulSize;
CK_RV rv;

.
.
rv = C_CreateObject(hSession, &template, 4, &hObject);
if (rv == CKR_OK) {
  rv = C_GetObjectSize(hSession, hObject, &ulSize);
  if (rv != CKR_INFORMATION_SENSITIVE) {
```

```
  .
  .
  }

  rv = C_DestroyObject(hSession, hObject);
  .
  .
}
```

♦ **C_GetAttributeValue**

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

C_GetAttributeValue получает значение одного или более атрибутов объекта; *hObject* – идентификатор объекта; *pTemplate* указывает на шаблон, в котором указывается, какие значения атрибутов необходимо извлекать, и получает значения атрибутов; *ulCount* – количество атрибутов в шаблоне.

Для каждой тройки (*type*, *pValue*, *ulValueLen*) в шаблоне **C_GetAttributeValue** выполняет следующий алгоритм:

1. Если указанный для объекта атрибут (т. е. атрибут, определенный полем *type*) не может открываться, так как объект является уязвимым или неизвлекаемым, поле *ulValueLen* в данной тройке будет модифицироваться под хранение значения минус 1 (т. е. когда оно приводится к CK_LONG, оно содержит минус 1).

2. В противном случае, если указанный атрибут для объекта является недействительным (объект не обладает таким атрибутом), поле *ulValueLen* в данной тройке будет модифицироваться под хранение значения минус 1.

3. В противном случае, если поле *pValue* имеет значение NULL_PTR, то поле *ulValueLen* модифицируется под хранение значения точной длины указанного атрибута объекта.

4. В противном случае, если длина, указанная в *ulValueLen*, является достаточно большой, для того чтобы хранить значение указанного атрибута для объекта, этот атрибут копируется в буфер, находящийся в *pValue*, а поле *ulValueLen* модифицируется под хранение значения точной длины атрибута.

5. В противном случае поле *ulValueLen* модифицируется под хранение значения минус 1.

Если случай 1 применим к любому из запрошенных атрибутов, вызов должен возвратить значение CKR_ATTRIBUTE_SENSITIVE. Если случай 2 применим к любому из запрошенных атрибутов, то вызов должен возвратить значение CKR_ATTRIBUTE_TYPE_INVALID. Если случай 5 применим к любому из запрошенных атрибутов, то вызов должен возвратить значение CKR_BUFFER_TOO_SMALL. Как обычно, если применим более чем один из таких кодов ошибок, то Cryptoki может выдать любой из них. Только если ни один из них неприменим ни к какому из запрошенных атрибутов, будет выдано значение CKR_OK.

В особом случае с атрибутом, значение которого состоит из массива атрибутов, например, **CKA_WRAP_TEMPLATE**, если оно передается при помощи поля *pValue*, значение которого не NULL, то, если значение поля *pValue* для элементов в пределах массива − NULL_PTR, то в поле *ulValueLen* элементов в пределах массива будет выставлена необходимая длина. Если значение поля *pValue* элементов в пределах массива не NULL_PTR, то элемент *ulValueLen* атрибутов в пределах массива должен отражать место, на которое указывает соответствующее поле *pValue*, а *pValue* заполняется, если есть достаточно места. Следовательно, важно инициализировать содержимое буфера до вызова **C_GetAttributeValue** для извлечения такого значения в виде массива. Если любой элемент *ulValueLen* в пределах массива недостаточно велик, он будет выставлен на минус 1, а функция возвратит CKR_BUFFER_TOO_SMALL так же, как она делает, если элемент *ulValueLen* аргумента *pTemplate* слишком мал. Обратите внимание, что любой атрибут, значение которого является массивом атрибутов, распознаваем по типу атрибута, в котором выставлен бит CKF_ARRAY_ATTRIBUTE.

Обратите внимание, что коды ошибок CKR_ATTRIBUTE_SENSITIVE, CKR_ATTRIBUTE_TYPE_ INVALID и CKR_BUFFER_TOO_SMALL не указывают на истинные ошибки функции **C_GetAttribute-Value**. Если вызов функции **C_GetAttributeValue** возвращает любое из этих трех значений, то данный вызов должен был обработать *каждый* из атрибутов в шаблоне, предоставленном функции **C_Get-AttributeValue**. Все атрибуты в шаблоне, чье значение *может* быть возвращено вызовом **C_GetAttribute-Value**, *будут* возвращены этим вызовом.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_SENSITIVE, CKR_ ATTRIBUTE_TYPE_INVALID, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_ DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_ GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_SESSION_ CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hObject;
CK_BYTE_PTR pModulus, pExponent;
CK_ATTRIBUTE template[] = {
 {CKA_MODULUS, NULL_PTR, 0},
 {CKA_PUBLIC_EXPONENT, NULL_PTR, 0}
};
CK_RV rv;

.
.
rv = C_GetAttributeValue(hSession, hObject, &template, 2);
if (rv == CKR_OK) {
 pModulus = (CK_BYTE_PTR) malloc(template[0].ulValueLen);
 template[0].pValue = pModulus;
 /* template[0].ulValueLen was set by C_GetAttributeValue */

 pExponent = (CK_BYTE_PTR) malloc(template[1].ulValueLen);
 template[1].pValue = pExponent;
 /* template[1].ulValueLen was set by C_GetAttributeValue */

 rv = C_GetAttributeValue(hSession, hObject, &template, 2);
 if (rv == CKR_OK) {
 .
 .
 }
 free(pModulus);
 free(pExponent);
}
```

- **C_SetAttributeValue**

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

C_SetAttributeValue изменяет значение одного или нескольких атрибутов объекта; *hSession* – идентификатор сеанса; *hObject* – идентификатор объекта; *pTemplate* указывает на шаблон, в котором указывается, какие значения атрибутов необходимо изменить, а также их новые значения; *ulCount* – количество атрибутов в шаблоне.

При ведении сеанса «только для чтения» можно изменять только объекты сеанса.

В шаблоне могут указываться новые значения для любых атрибутов объекта, который может быть изменен. Если в шаблоне указано значение атрибута, которое несовместимо с существующими атрибутами объекта, то вызов завершится сбоем с кодом ошибки CKR_TEMPLATE_INCONSISTENT.

Не все атрибуты можно изменять, для получения более подробной информации см. 9.7.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OBJECT_HANDLE_INVALID, CKR_OK, CKR_ SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_ INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN.

Пример:

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

.
.
rv = C_SetAttributeValue(hSession, hObject, &template, 1);
if (rv == CKR_OK) {
  .
  .
}
```

♦ **C_FindObjectsInit**

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

C_FindObjectsInit инициализирует поиск объектов носителя и сеанса, которые подходят шаблону. *hSession* – идентификатор сеанса; *pTemplate* указывает на шаблон поиска, в котором приводятся необходимые значения атрибутов; *ulCount* – количество атрибутов в шаблоне поиска. Критерий совпадения – это точное совпадение байт в байт со всеми атрибутами шаблона. Чтобы найти все объекты, выставьте *ulCount* на 0.

После вызова функции **C_FindObjectsInit** приложение может вызвать **C_FindObjects** один или несколько раз для извлечения идентификаторов объектов, совпадающих с объектами шаблона, и затем вызвать **C_FindObjectsFinal** для завершения активной операции поиска. В большинстве случаев операция поиска может быть активной в определенное время в определенном сеансе.

Операция поиска объекта будет находить только объекты, которые может просматривать сеанс. Например, операция поиска объекта в «открытом сеансе чтения/записи» не найдет никаких секретных объектов (даже если один из атрибутов в шаблоне поиска указывает, что идет поиск секретных объектов).

Если операция поиска является активной, а объекты создаются или уничтожаются, что соответствует шаблону для активной поисковой операции, эта операция может найти, а может и не найти такие объекты. Обратите внимание, что операция поиска может выдать недействительные идентификаторы объектов.

Даже несмотря на то, что функция **C_FindObjectsInit** может выдать значения CKR_ATTRIBUTE_ TYPE_INVALID и CKR_ATTRIBUTE_VALUE_INVALID, это не требуется безусловно. Например, если функция предоставляет шаблон поиска с несуществующими атрибутами, она может выдать CKR_ ATTRIBUTE_TYPE_INVALID либо инициализировать операцию поиска, которая не найдет совпадающих объектов и выдаст CKR_OK.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ ATTRIBUTE_VALUE_INVALID, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_ MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_ MEMORY, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_ SESSION_HANDLE_INVALID.

Пример: см. C_FindObjectsFinal.

♦ **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 продолжает поиск объектов носителя и сеанса, которые подходили бы к шаблону, получая информацию о дополнительных идентификаторах объектов. *hSession* – идентификатор сеанса; *phObject* указывает на адрес участка памяти, который получает список (массив) дополнительных идентификаторов объектов; *ulMaxObjectCount* – максимальное количество идентификаторов объектов, которые могут быть выданы; *pulObjectCount* указывает на адрес участка памяти, который получает информацию о текущем количестве выдаваемых идентификаторов объектов.

Если больше нет объектов, которые подходят в шаблон, то в участок памяти, на который указывает *pulObjectCount*, поступает значение 0.

Поиск должен инициализироваться с использованием функции **C_FindObjectsInit**.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_ DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_ GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_ SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример: см. C_FindObjectsFinal.

♦ **C_FindObjectsFinal**

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

C_FindObjectsFinal прекращает операцию поиска объектов носителя и сеанса. *hSession* – идентификатор сеанса.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_ DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_ HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_ SESSION_HANDLE_INVALID.

Пример:

```
CK_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);
```

11.8 Функции шифрования

Cryptoki предоставляет следующие функции для шифрования данных:

♦ **C_EncryptInit**

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

C_EncryptInit инициализирует операцию шифрования. *hSession* − идентификатор сеанса; *pMechanism* указывает на механизм шифрования; *hKey* − идентификатор ключа шифрования.

Атрибут **CKA_ENCRYPT** ключа шифрования, который указывает, поддерживает ли данный ключ шифрование; если да, он должен быть выставлен на CK_TRUE.

После вызова **C_EncryptInit** приложение может либо вызвать **C_Encrypt** для шифрования данных в одной части, либо вызвать **C_EncryptUpdate** ноль или более раз с последующим вызовом **C_Encrypt-Final** для шифрования данных в нескольких частях. Операция шифрования будет активной, пока приложение использует вызов **C_Encrypt** или **C_EncryptFinal,** *чтобы фактически получить* последнюю часть шифртекста. Для обработки дополнительных данных (в одной или нескольких частях) приложение должно снова вызвать **C_EncryptInit**.

Возвращаемые значения: 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.

Пример: см. C_EncryptFinal.

- **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 шифрует односторонние данные. *hSession* – идентификатор сеанса; *pData* указывает на данные; *ulDataLen* – длина данных в байтах; *pEncryptedData* указывает на адрес участка памяти, в который передаются зашифрованные данные; *pulEncryptedDataLen* указывает на адрес участка памяти, в котором содержится информация о длине шифрованных данных в байтах.

Производя вывод данных, **C_Encrypt** придерживается норм, описанных в 11.2.

Операция шифрования должна быть инициализирована с использованием функции **C_EncryptInit**. Вызов **C_Encrypt** всегда прекращает активную операцию шифрования, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения шифртекста.

Функция **C_Encrypt** не может использоваться для прекращения многосторонней операции и должна вызываться после вызова **C_EncryptInit** без промежуточных вызовов **C_EncryptUpdate**.

В некоторых механизмах шифрования ввод данных открытого текста имеет определенные ограничения по длине (либо в связи с тем, что механизм может шифровать только относительно короткие части открытого текста, либо из-за того, что для того или иного механизма вводимый массив должен состоять из целого числа блоков). Если эти ограничения не соблюдаются, то функция **C_Encrypt** завершится сбоем с кодом ошибки CKR_DATA_LEN_RANGE.

И открытый, и шифрованный тексты могут находиться в одном и том же месте, т. е. *pData* и *pEncryptedData* могут указывать на одну и ту же позицию.

В большинстве механизмов **C_Encrypt** является эквивалентом последовательности операций **C_EncryptUpdate** с последующим выполнением **C_EncryptFinal**.

Возвращаемые значения: 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.

Пример: для ознакомления с примерами подобных функций см. C_EncryptFinal.

- **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 продолжает операцию многостороннего шифрования, обрабатывая еще одну часть данных. *hSession* – идентификатор сеанса; *pPart* указывает на часть данных; *ulPartLen* – длина части данных; *pEncryptedPart* указывает на адрес участка памяти, в которую передается шифрованная часть данных; *pulEncryptedDataLen* указывает на адрес участка памяти, в котором содержится информация о длине шифрованной части данных в байтах.

Производя вывод данных, **C_EncryptUpdate** придерживается норм, описанных в 11.2.

Операция шифрования должна быть инициализирована с использованием функции **C_EncryptInit**. Эта функция может вызваться последовательно несколько раз. Вызов **C_Encrypt-Update**, который выдает код ошибки, отличающийся от CKR_BUFFER_TOO_SMALL, прекращает текущую операцию шифрования.

И открытый текст, и шифртекст могут находиться в одном и том же месте, т. е. *pPart* и *pEncrypted-Part* могут указывать на одну и ту же позицию.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_ CRYPTOKI_NOT_INITIALIZED, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_ CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример: см. C_EncryptFinal.

- **C_EncryptFinal**

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

C_EncryptFinal завершает операцию многостороннего шифрования. *hSession* – идентификатор сеанса; *pLastEncryptedPart* указывает на адрес участка памяти, в который поступает последняя часть шифрованных данных, если таковая существует; *pulLastEncryptedPartLen* указывает на адрес участка памяти, в котором хранится информация о длине последней части зашифрованных данных.

Производя вывод данных, **C_EncryptFinal** придерживается норм, описанных в 11.2.

Операция шифрования должна быть инициализирована с использованием функции **C_EncryptInit**. Вызов **C_EncryptFinal** всегда прекращает активную операцию шифрования, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения шифртекста.

В некоторых механизмах многостороннего шифрования ввод данных открытого текста имеет определенные ограничения по длине из-за того, что для того или иного механизма вводимый массив должен состоять из целого числа блоков. Если эти ограничения не соблюдаются, то функция **C_Encrypt** завершится сбоем с кодом ошибки CKR_DATA_LEN_RANGE.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_ CRYPTOKI_NOT_INITIALIZED, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример:

```
#define PLAINTEXT_BUF_SZ 200
#define CIPHERTEXT_BUF_SZ 256

CK_ULONG firstPieceLen, secondPieceLen;
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_BYTE iv[8];
CK_MECHANISM mechanism = {
 CKM_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) {
  .
  .
  }
}
```

11.9 Функции расшифрования

Cryptoki предоставляет следующие функции для расшифрования данных:

♦ **C_DecryptInit**

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

C_DecryptInit инициализирует операцию расшифрования. *hSession* − идентификатор сеанса; *pMechanism* указывает на механизм расшифрования; *hKey* − идентификатор ключа расшифрования.

Атрибут **CKA_DECRYPT** ключа расшифрования, который указывает, поддерживает ли данный ключ расшифрование; если да, он должен быть выставлен на CK_TRUE.

После вызова **C_DecryptInit** приложение может либо вызвать **C_Decrypt** для расшифрования данных в одной части, либо вызвать **C_DecryptUpdate** ноль или более раз с последующим вызовом **C_DecryptFinal** для расшифрования данных в нескольких частях. Операция расшифрования будет активной, пока приложение использует вызов **C_Decrypt** или **C_DecryptFinal,** *чтобы фактически получить* последнюю часть открытого текста. Для обработки дополнительных данных (в одной или нескольких частях) приложение должно снова вызвать **C_DecryptInit**.

Возвращаемые функцией значения: 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.

Пример: см. C_DecryptFinal.

♦ **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 расшифровывает зашифрованные данные в одной части. *hSession* – идентификатор сеанса; *pEncryptedData* указывает на зашифрованные данные; *ulEncryptedDataLen* – длина зашифрованных данных; *pData* указывает на адрес участка памяти, в котором хранятся расшифрованные данные; *pulDataLen* указывает на адрес участка памяти, в котором хранится длина расшифрованных данных.

Производя вывод данных, **C_Decrypt** придерживается норм, описанных в 11.2.

Операция расшифрования должна быть инициализирована с использованием функции **C_DecryptInit**. Вызов **C_Decrypt** всегда прекращает активную операцию расшифрования, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения открытого текста.

Функция **C_Decrypt** не может использоваться для прекращения многосторонней операции и должна вызываться после вызова **C_DecryptInit** без промежуточных вызовов **C_DecryptUpdate**.

И открытый текст, и шифртекст могут находиться в одном и том же месте, т. е. *pEncryptedData* и *pData* указывают на одну и ту же позицию.

Если введенные данные шифртекста невозможно расшифровать из-за их ненадлежащей длины, будет выдан код ошибки CKR_ENCRYPTED_DATA_INVALID или CKR_ENCRYPTED_DATA_LEN_RANGE.

В большинстве механизмов **C_Decrypt** – это эквивалент последовательности операций **C_Decrypt-Update** с последующим выполнением **C_DecryptFinal**.

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

Пример: для ознакомления с примерами подобных функций см. C_DecryptFinal.

- **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 продолжает операцию многостороннего расшифрования, обрабатывая еще одну часть зашифрованных данных. *hSession* – идентификатор сеанса; *pEncryptedPart* указывает на адрес участка памяти, в который поступает зашифрованная часть данных; *ulEncryptedPartLen* – длина части зашифрованных данных; *pPart* указывает на адрес участка памяти, в который поступают расшифрованные данные; *pulPartLen* указывает на адрес участка памяти, в котором содержится информация о длине расшифрованной части данных.

Производя вывод данных, **C_DecryptUpdate** придерживается норм, описанных в 11.2.

Операция расшифрования должна быть инициализирована с использованием функции **C_DecryptInit**. Эта функция может вызваться последовательно несколько раз. Вызов **C_DecryptUpdate**, который выдает код ошибки, отличающийся от CKR_BUFFER_TOO_SMALL, прекращает текущую операцию расшифрования.

И открытый текст, и шифртекст могут находиться в одном и том же месте, т. е. *pEncryptedPart* и *pPart* указывают на одну и ту же позицию.

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

Пример: см. C_DecryptFinal.

- **C_DecryptFinal**

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

C_DecryptFinal завершает операцию многостороннего расшифрования. *hSession* – идентификатор сеанса; *pLastPart* указывает на адрес участка памяти, в который передается последняя часть расшифрованных данных, если таковая существует; *pulLastPartLen* указывает на адрес участка памяти, в котором хранится информация о длине последней части расшифрованных данных.

Производя вывод данных, **C_DecryptFinal** придерживается норм, описанных в 11.2.

Операция расшифрования должна быть инициализирована с использованием функции **C_DecryptInit**. Вызов **C_DecryptFinal** всегда прекращает активную операцию расшифрования, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения открытого текста.

Если введенные данные шифртекста невозможно расшифровать из-за их ненадлежащей длины, будет выдан код ошибки CKR_ENCRYPTED_DATA_INVALID или CKR_ENCRYPTED_DATA_LEN_RANGE.

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

Пример:

```
#define CIPHERTEXT_BUF_SZ 256
#define PLAINTEXT_BUF_SZ 256

CK_ULONG firstEncryptedPieceLen, secondEncryptedPieceLen;
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_BYTE iv[8];
CK_MECHANISM mechanism = {
 CKM_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) {
  .
  .
  }
}
```

11.10 Функции получения списка данных

Cryptoki предоставляет следующие функции для получения списка данных:

♦ **C_DigestInit**

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

C_DigestInit инициализирует операцию получения списка сообщения. *hSession* – идентификатор сеанса; *pMechanism* указывает на механизм получения списка.

После вызова **C_DigestInit** приложение может либо вызвать **C_Digest** для получения списка одной части текста, либо вызвать **C_DigestUpdate** ноль или более раз, после чего вызвать **C_Digest-Final** для получения списка нескольких частей. Операция получения списка будет активной, пока приложение использует вызов **C_Digest** или **C_DigestFinal,** *чтобы фактически получить* список сообщения. Для обработки дополнительных данных (в одной или нескольких частях) приложение должно снова вызвать **C_DigestInit**.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_ DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_ FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_MECHANISM_INVALID, CKR_ MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_ SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

Пример: см. 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_Decrypt вычисляет список данных одной части обрабатываемого сообщения. *hSession* – идентификатор сеанса; *pData* указывает на данные; *ulDataLen* – длина данных; *pDigest* указывает на адрес участка памяти, в который передается список; *pulDigestLen* указывает на адрес участка памяти, в котором хранится информация о длине списка сообщения.

Производя вывод данных, **C_Digest** придерживается норм, описанных в 11.2.

Операция получения списка данных должна быть инициализирована с использованием функции **C_DigestInit**. Вызов **C_Digest** всегда прекращает активную операцию получения списка, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения списка сообщения.

Функция **C_Digest** не может использоваться для прекращения многосторонней операции и должна вызываться после вызова **C_DigestInit** без промежуточных вызовов **C_DigestUpdate**.

И открытый текст, и шифртекст могут находиться в одном и том же месте, т. е. *pData* и *pDigest* указывают на одну и ту же позицию.

C_Digest – это эквивалент последовательности операций **C_DigestUpdate** с последующим выполнением **C_DigestFinal.**

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_ NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_ FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример: для ознакомления с примерами подобных функций см. C_DigestFinal.

♦ **C_DigestUpdate**

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

C_DigestUpdate продолжает многостороннюю операцию получения списка данных, обрабатывая еще одну часть данных. *hSession* – идентификатор сеанса, *pPart* указывает на часть данных; *ulPartLen* – длина части данных.

Операция получения списка данных должна быть инициализирована с использованием функции **C_DigestInit**. Вызовы данной функции и функции **C_DigestKey** могут чередоваться любое количество раз в любом порядке. Вызов **C_DigestUpdate**, который приводит к ошибке, прекращает текущую операцию получения списка.

Возвращаемые значения: 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.

Пример: см. C_DigestFinal.

- **C_DigestKey**

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

C_DigestKey продолжает многостороннюю операцию получения списка данных путем вычисления списка значения секретного ключа. *hSession* – идентификатор сеанса; *hKey* – это идентификатор секретного ключа, который должен быть подвергнут операции получения списка.

Операция получения списка данных должна быть инициализирована с использованием функции **C_DigestInit**. Вызовы данной функции и функции **C_DigestKey** могут чередоваться любое количество раз в любом порядке.

Если значение предоставленного ключа не может использоваться в операции получения списка данных по каким-либо причинам, связанным с его длиной, функция **C_DigestKey** должна выдать код ошибки CKR_KEY_SIZE_RANGE.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_ MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_ GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_HANDLE_INVALID, CKR_KEY_INDIGESTIBLE, CKR_ KEY_SIZE_RANGE, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_ SESSION_HANDLE_INVALID.

Пример: см. C_DigestFinal.

- **C_DigestFinal**

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

C_DigestFinal завершает многостороннюю операцию получения списка сообщения, выдавая список; *hSession* – идентификатор сеанса; *pDigest* указывает на адрес участка памяти, в который передается значение списка; *pulDigestLen* указывает на адрес участка памяти, в котором хранится длина списка.

Производя вывод данных, **C_DigestFinal** придерживается норм, описанных в 11.2.

Операция получения списка данных должна быть инициализирована с использованием функции **C_DigestInit**. Вызов **C_Digest** всегда прекращает активную операцию получения списка, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения списка сообщения.

Возвращаемые функцией значения: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_MECHANISM mechanism = {
  CKM_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);
.
.
```

11.11 Функции электронной подписи и MAC

Ниже описываются функции, предоставляемые Cryptoki для заверения данных электронной подписью (при использовании для Cryptoki эти функции охватывают также коды аутентификации сообщений (Message Authentification Code, MAC):

♦ **C_SignInit**

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

C_SignInit инициализирует операцию электронной подписи, при этом подпись является приложением к данным. *hSession* – идентификатор сеанса; *pMechanism* указывает на механизм подписи; *hKey* – идентификатор ключа подписи.

Атрибут **CKA_SIGN** ключа подписи, который указывает, поддерживает ли данный ключ подписи с приложением; если да, то он должен быть выставлен на CK_TRUE.

После вызова **C_SignInit** приложение может либо вызвать **C_Sign** для подписи данных одной части текста, либо вызвать **C_SignUpdate** один или более раз, после чего вызвать **C_SignFinal** для подписи данных нескольких частей. Операция постановки электронной подписи будет активной, пока приложение использует вызов **C_Sign** или **C_SignFinal,** *чтобы фактически получить* подпись. Для обработки дополнительных данных (в одной или нескольких частях) приложение должно снова вызвать **C_SignInit**.

Возвращаемые значения: 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.

Пример: см. C_SignFinal.

♦ **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 заверяет электронной подписью одну часть данных, при этом подпись является приложением к данным. *hSession* – идентификатор сеанса; *pData* указывает на данные; *ulDataLen* – длина данных; *pSignature* указывает на адрес участка памяти, в который передается подпись; *pulSignatureLen* указывает на адрес участка памяти, в котором хранится информация о длине подписи.

Производя вывод данных, **C_Sign** придерживается норм, описанных в 11.2.

Операция постановки электронной подписи должна быть инициализирована с использованием функции **C_SignInit**. Вызов **C_SignFinal** всегда прекращает активную операцию постановки электронной подписи, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения подписи.

Функция **C_Sign** не может использоваться для прекращения многосторонней операции и должна вызываться после вызова **C_SignInit** без промежуточных вызовов **C_SignUpdate**.

В большинстве механизмов **C_Sign** – это эквивалент последовательности операций **C_SignUpdate** с последующим выполнением **C_SignFinal**.

Возвращаемые значения: 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.

Пример: для ознакомления с примерами подобных функций см. C_SignFinal.

♦ **C_SignUpdate**

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

C_SignUpdate продолжает многостороннюю операцию постановки электронной подписи, заверяя электронной подписью одну часть данных. *hSession* – идентификатор сеанса; *pPart* указывает на элемент данных; *ulPartLen* – длина элемента данных.

Операция постановки электронной подписи должна быть инициализирована с использованием функции **C_SignInit**. Вызовы данной функции могут последовательно производиться любое количество раз. Вызов **C_SignUpdate**, который приводит к ошибке, прекращает текущую операцию постановки электронной подписи.

Возвращаемые значения: 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.

Пример: см. C_SignFinal.

♦ **C_SignFinal**

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

C_SignFinal завершает многостороннюю операцию постановки электронной подписи, выдавая электронную подпись. *hSession* – идентификатор сеанса; *pSignature* указывает на адрес участка памяти, в который передается электронная подпись; *pulSignatureLen* указывает на адрес участка памяти, в котором хранится информация о длине подписи.

Производя вывод данных, **C_SignFinal** придерживается норм, описанных в 11.2.

Операция постановки электронной подписи должна быть инициализирована с использованием функции **C_SignInit**. Вызов **C_SignFinal** всегда прекращает активную операцию постановки электронной подписи, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения подписи.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_ NOT_INITIALIZED, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_ REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_ MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_ HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN, CKR_FUNCTION_REJECTED.

Пример:

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

♦ **C_SignRecoverInit**

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

C_SignRecoverInit инициализирует операцию подписи, при которой данные могут быть восстановлены из подписи. *hSession* – идентификатор сеанса; *pMechanism* указывает на механизм подписи; *hKey* – идентификатор ключа подписи.

Атрибут **CKA_SIGN_RECOVER** ключа подписи, который указывает, поддерживает ли данный ключ подписи, из которых могут восстанавливаться данные; должен быть выставлен на CK_TRUE.

После вызова **C_SignRecoverInit** приложение может вызвать **C_SignRecover** для заверки подписью одной части. Операция постановки электронной подписи будет активной, пока приложение использу-

ет вызов **C_SignRecover,** *чтобы фактически получить* подпись. Для обработки дополнительных данных приложение должно снова вызвать **C_SignRecoverInit**.

Возвращаемые значения: 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.

Пример: см. C_SignRecover.

♦ **C_SignRecover**

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

C_SignRecover заверяет данные электронной подписью в ходе единой операции, при этой операции данные могут быть восстановлены из подписи. *hSession* – идентификатор сеанса; *pData* указывает на данные; *ulDataLen* – длина данных; *pSignature* указывает на адрес участка памяти, в который помещается подпись; *pulSignatureLen* указывает на адрес участка памяти, в котором хранится информация о длине подписи.

Производя вывод данных, **C_SignRecover** придерживается норм, описанных в 11.2.

Операция постановки электронной подписи должна быть инициализирована с использованием функции **C_SignRecoverInit**. Вызов **C_SignRecover** всегда прекращает активную операцию постановки электронной подписи, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения подписи.

Возвращаемые значения: 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.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_MECHANISM mechanism = {
 CKM_RSA_9796, NULL_PTR, 0
};
CK_BYTE data[] = {...};
CK_BYTE signature[128];
CK_ULONG ulSignatureLen;
CK_RV rv;

.
.
rv = C_SignRecoverInit(hSession, &mechanism, hKey);
if (rv == CKR_OK) {
  ulSignatureLen = sizeof(signature);
  rv = C_SignRecover(
  hSession, data, sizeof(data), signature, &ulSignatureLen);
  if (rv == CKR_OK) {
  .
  .
  }
}
```

11.12 Функции верификации подписей и MAC

Ниже описываются функции, предоставляемые Cryptoki для верификации заверки данных электронной подписью (при использовании для Cryptoki эти функции охватывают также коды аутентификации сообщений (Message Authentification Code, MAC):

♦ **C_VerifyInit**

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

C_VerifyInit инициализирует операцию верификации подписи, где подпись является приложением к данным. *hSession* – идентификатор сеанса; *pMechanism* указывает на механизм верификации подписи; *hKey* – идентификатор верификационного ключа.

Атрибут **CKA_VERIFY** верификационного ключа, который указывает, поддерживает ли данный ключ операцию верификации подписи, где подпись является приложением к данным, должен быть выставлен на CK_TRUE.

После вызова **C_VerifyInit** приложение может либо вызвать **C_Verify** для подписи данных одной части текста, либо вызвать **C_VerifyUpdate** один или более раз, после чего вызвать _**VerifyFinal** для подписи данных нескольких частей. Операция постановки электронной подписи будет активной, пока приложение использует вызов **C_Verify** или **C_VerifyFinal**. Для обработки дополнительных данных (в одной или нескольких частях) приложение должно снова вызвать **C_VerifyInit**.

Возвращаемые значения: 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.

Пример: см. C_VerifyFinal.

♦ **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 верифицирует подпись в ходе единой операции, где подпись является приложением к данным. *hSession* – идентификатор сеанса; *pData* указывает на данные; *ulDataLen* – длина данных; *pSignature* указывает на длину подписи.

Операция верификации должна быть инициализирована с использованием функции **C_VerifyInit**. Вызов **C_Verify** всегда прекращает текущую операцию верификации.

Успешный вызов **C_Verify** должен привести к выдаче значения CKR_OK (указывающего, что предоставленная подпись является действительной) или значения CKR_SIGNATURE_INVALID (указывающего, что предоставленная подпись является недействительной). Если подпись рассматривается как недействительная, исходя только из ее длины, то будет выдано значение CKR_SIGNATURE_LEN_RANGE. В любом из этих случаев активная операция подписи прерывается.

Функция **C_Verify** не может использоваться для прекращения многосторонней операции и должна вызываться после вызова **C_VerifyInit** без промежуточных вызовов **C_VerifyUpdate**.

В большинстве механизмов **C_Verify** – эквивалент последовательности операций **C_VerifyUpdate** с последующим выполнением **C_VerifyFinal**.

Возвращаемые значения: 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.

Пример: для ознакомления с примерами подобных функций см. C_VerifyFinal.

- **C_VerifyUpdate**

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

C_VerifyUpdate продолжает многостороннюю операцию подписи, обрабатывая еще одну часть данных. *hSession* – идентификатор сеанса; *pPart* указывает на элемент данных; *ulPartLen* – длина элемента данных.

Операция верификации должна быть инициализирована с использованием функции **C_VerifyInit**. Вызовы данной функции могут последовательно производиться любое количество раз. Вызов **C_SignUpdate**, который приводит к ошибке, прекращает текущую операцию верификации.

Возвращаемые значения: 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.

Пример: см. C_VerifyFinal.

- **C_VerifyFinal**

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

C_VerifyFinal завершает многостороннюю операцию верификации, проверяя подпись. *hSession* – идентификатор сеанса; *pSignature* указывает на подпись; *ulSignatureLen* – длина подписи.

Операция верификации должна была инициализироваться с использованием функции **C_VerifyInit**. Вызов **C_VerifyFinal** всегда останавливает текущую операцию верификации.

Успешный вызов **C_VerifyFinal** должен привести к выдаче значения CKR_OK (указывающего, что предоставленная подпись является действительной) или значения CKR_SIGNATURE_INVALID (указывающего, что предоставленная подпись является недействительной). Если подпись рассматривается как недействительная, исходя только из ее длины, то будет выдано значение CKR_SIGNATURE_ LEN_RANGE. В любом из этих случаев активная операция подписи прерывается.

Возвращаемые функцией значения: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SIGNATURE_INVALID, CKR_SIGNATURE_LEN_RANGE.

Пример:

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

♦ **C_VerifyRecoverInit**

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

C_VerifyRecoverInit инициализирует операцию верификации подписи, при этой операции данные могут быть восстановлены из подписи. *hSession* – идентификатор сеанса; *pMechanism* указывает на структуру, задающую механизм верификации; *hKey* – идентификатор ключа верификации.

Атрибут **CKA_VERIFY_RECOVER** верификационного ключа, который указывает, поддерживает ли данный ключ операцию верификации подписи, где данные могут быть восстановлены из подписи, должен быть выставлен на CK_TRUE.

После вызова **C_VerifyRecoverInit** приложение может вызвать **C_VerifyRecover** для верификации подписи одной части данных. Операция верификации электронной подписи будет активной, пока приложение использует вызов **C_VerifyRecover,** *чтобы фактически получить* восстановленные данные.

Возвращаемые значения: 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.

Пример: см. C_VerifyRecover.

♦ **C_VerifyRecover**

```
CK_DEFINE_FUNCTION(CK_RV, C_VerifyRecover)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pSignature,
  CK_ULONG ulSignatureLen,
  CK_BYTE_PTR pData,
  CK_ULONG_PTR pulDataLen
);
```

C_VerifyRecover выполняет верификацию электронной подписи в ходе единой операции, где данные могут быть восстановлены из подписи. *hSession* – идентификатор сеанса; *pSignature* указывает на подпись; *ulSignatureLen* – длина подписи; *pData* указывает на адрес участка памяти, в который поступают восстановленные данные; *pulDataLen* указывает на адрес участка памяти, в котором хранятся извлеченные данные.

Производя вывод данных, **C_VerifyRecover** придерживается норм, описанных в 11.2.

Операция верификации должна была инициализироваться с использованием функции **C_VerifyRecoverInit**. Вызов **C_VerifyRecover** всегда прекращает активную операцию верификации подписи, если он не возвращает CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тот, который возвращает CKR_OK), чтобы определить длину буфера, который нужен для хранения извлеченных данных.

Операция верификации должна быть инициализирована с использованием функции **C_VerifyRecoverInit**. Вызов **C_VerifyRecover** всегда прекращает активную операцию верификации электронной подписи, если он не выдает значение CKR_BUFFER_TOO_SMALL или не является успешным вызовом (т. е. тем, который выдает CKR_OK), чтобы определить длину буфера, который нужен для хранения восстановленных данных.

Успешный вызов **C_VerifyRecover** должен привести к выдаче значения CKR_OK (указывающего, что предоставленная подпись является действительной) или значения CKR_SIGNATURE_INVALID (указывающего, что предоставленная подпись является недействительной). Если подпись рассматривается как недействительная, только исходя из ее длины, будет выдано значение CKR_SIGNATURE_ LEN_RANGE. Возвращаемые коды CKR_SIGNATURE_INVALID и CKR_SIGNATURE_LEN_RANGE имеют больший приоритет, чем возвращаемый код ошибки CKR_BUFFER_TOO_SMALL, т. е. если функции **C_VerifyRecover** предоставляется недействительная подпись, она никогда не выдаст CKR_ BUFFER_TOO_SMALL.

Возвращаемые значения: 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_SIGNATURE_LEN_RANGE, CKR_SIGNATURE_INVALID.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_MECHANISM mechanism = {
  CKM_RSA_9796, NULL_PTR, 0
};
CK_BYTE data[] = {...};
CK_ULONG ulDataLen;
CK_BYTE signature[128];
CK_RV rv;

.
.
rv = C_VerifyRecoverInit(hSession, &mechanism, hKey);
if (rv == CKR_OK) {
  ulDataLen = sizeof(data);
  rv = C_VerifyRecover(
  hSession, signature, sizeof(signature), data, &ulDataLen);
  .
  .
}
```

11.13 Криптографические функции двойного действия

Ниже описываются функции, предоставляемые Cryptoki для выполнения двух криптографических операций «одновременно» в пределах сеанса. Данные функции предназначены для избежания ненужной передачи данных туда и обратно – на носитель и из носителя.

- **C_DigestEncryptUpdate**

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

C_DigestEncryptUpdate продолжает многосторонние операции получения списка и шифрования, обрабатывая еще одну часть данных. *hSession* – идентификатор сеанса; *pPart* указывает на элемент данных; *ulPartLen* – это длина элемента данных; *pEncryptedPart* указывает на адрес участка памяти, в которую передается часть данных после получения списка и шифрования; *pulEncryptedPartLen* указывает на адрес участка памяти, который содержит информацию о длине шифрованной части данных.

Производя вывод данных, **C_DigestEncryptUpdate** придерживается норм, описанных в 11.2. Если вызов **C_DigestEncryptUpdate** не выдает зашифрованных данных (что обусловлено ошибками или тем, что *pEncryptedPart* имеет значение NULL_PTR, либо недостаточным размером поля *pulEncrypted-PartLen*, чтобы содержать всю выводимую часть шифрованных данных), то никакой открытый текст не передается активной операции получения списка.

Обе операции получения списка и шифрования должны быть активными (они должны быть инициализированы функциями **C_DigestInit** и **C_EncryptInit** соответственно). Данная функция может вызываться любое количество раз подряд, а также может чередоваться с вызовами функций **C_DigestUpdate**, **C_DigestKey** и **C_EncryptUpdate** (однако было бы несколько необычно, если бы вызовы **C_DigestEncryptUpdate** чередовались с вызовами **C_DigestKey**).

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_ NOT_INITIALIZED, CKR_DATA_LEN_RANGE, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_ REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_ MEMORY, CKR_OK, CKR_OPERATION_NOT_INITIALIZED, CKR_SESSION_CLOSED, CKR_SESSION_ HANDLE_INVALID.

Пример:

```
#define BUF_SZ 512

CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_BYTE iv[8];
CK_MECHANISM digestMechanism = {
  CKM_MD5, NULL_PTR, 0
};
CK_MECHANISM encryptionMechanism = {
  CKM_DES_ECB, iv, sizeof(iv)
};
CK_BYTE encryptedData[BUF_SZ];
CK_ULONG ulEncryptedDataLen;
CK_BYTE digest[16];
CK_ULONG ulDigestLen;
CK_BYTE data[(2*BUF_SZ)+8];
CK_RV rv;
int i;

.
.
memset(iv, 0, sizeof(iv));
memset(data, 'A', ((2*BUF_SZ)+5));
rv = C_EncryptInit(hSession, &encryptionMechanism, hKey);
if (rv != CKR_OK) {
  .
  .
}
rv = C_DigestInit(hSession, &digestMechanism);
if (rv != CKR_OK) {
  .
  .
}

ulEncryptedDataLen = sizeof(encryptedData);
rv = C_DigestEncryptUpdate(
  hSession,
  &data[0], BUF_SZ,
  encryptedData, &ulEncryptedDataLen);
.
.
ulEncryptedDataLen = sizeof(encryptedData);
rv = C_DigestEncryptUpdate(
  hSession,
  &data[BUF_SZ], BUF_SZ,

  encryptedData, &ulEncryptedDataLen);
.
.


/*
 * The last portion of the buffer needs to be handled with
```

```
 * separate calls to deal with padding issues in ECB mode
 */

/* First, complete the digest on the buffer */
rv = C_DigestUpdate(hSession, &data[BUF_SZ*2], 5);
.
.
ulDigestLen = sizeof(digest);
rv = C_DigestFinal(hSession, digest, &ulDigestLen);
.
.

/* Then, pad last part with 3 0x00 bytes, and complete encryption */
for(i=0;i<3;i++)
  data[((BUF_SZ*2)+5)+i] = 0x00;

/* Now, get second-to-last piece of ciphertext */
ulEncryptedDataLen = sizeof(encryptedData);
rv = C_EncryptUpdate(
  hSession,
  &data[BUF_SZ*2], 8,
  encryptedData, &ulEncryptedDataLen);
.
.

/* Get last piece of ciphertext (should have length 0, here) */
ulEncryptedDataLen = sizeof(encryptedData);
rv = C_EncryptFinal(hSession, encryptedData, &ulEncryptedDataLen);
.
.
```

♦ **C_DecryptDigestUpdate**

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

C_DecryptDigestUpdate продолжает многостороннюю объединенную операцию расшифрования и получения списка, обрабатывая еще одну часть данных. *hSession* – идентификатор сеанса; *pEncryptedPart* указывает на зашифрованный элемент данных; *ulEncryptedPartLen* – длина элемента зашифрованных данных; *pPart* указывает на адрес участка памяти, в который передается расшифрованный элемент данных; *pulPartLen* указывает на адрес участка памяти, в котором содержится информация о длине восстановленного элемента данных.

Производя вывод данных, **C_DecryptDigestUpdate** придерживается норм, описанных в 11.2. Если вызов **C_DecryptDigestUpdate** не выдает результирующего вывода расшифрованных данных (из-за ошибок, или того, что *pPart* имеет значение NULL_PTR, или потому, что размер поля *pulPartLen* недостаточен, чтобы содержать всю выдаваемую часть расшифрованных данных), то никакой открытый текст не передается активной операции получения списка.

Обе операции получения списка и шифрования должны быть активными (они должны быть инициализированы функциями **C_DecryptInit** и **C_DigestInit** соответственно). Данная функция может вызываться любое число раз подряд, а также может чередоваться с вызовами функций **C_DecryptUpdate**, **C_DigestUpdate** и **C_DigestKey** (однако было бы несколько необычно, если бы вызовы **C_DigestEncryptUpdate** чередовались с вызовами **C_DigestKey**).

При использовании функции **C_DecryptDigestUpdate** появляется вопрос сцепления [конвейеризации], который не возникает при использовании **C_DigestEncryptUpdate**, «обратной функции» **C_DecryptDigestUpdate**. Это происходит потому, что при вызове **C_DigestEncryptUpdate** абсолютно одинаковые данные ввода передаются как активной операции получения списка, так и активной операции шифрования; однако, когда вызывается **C_DecryptDigestUpdate**, данные ввода, передаваемые активной операции получения списка, являются *данными вывода* активной операции расшифрования. Этот

вопрос возникает только когда механизм, применяемый для расшифрования, использует заполнение.

Представим, в частности, 24-байтный шифртекст, который был получен посредством шифрования 18-байтного открытого текста алгоритмом DES в режиме CBC с заполнением PKCS. Рассмотрим приложение, которое будет одновременно расшифровывать этот шифртекст и вычислять список получаемого таким образом исходного открытого текста.

После инициализации операций расшифрования и списка приложение передает 24-байтный шифртекст (3 блока DES) функции **C_DecryptDigestUpdate**. **C_DecryptDigestUpdate** возвращает ровно 16 байт открытого текста, так как в данный момент Cryptoki не знает, последует ли еще порция шифртекста и содержит ли последний блок шифртекста какое-либо заполнение. Эти 16 байт открытого текста передаются активной операции получения списка.

Так как шифртекста больше нет, приложение вызывает **C_DecryptFinal**. Тем самым Cryptoki уведомляется, что новая порция шифртекста не последует, и результатом данного вызова становятся последние 2 байта открытого текста. Однако, поскольку активные операции расшифрования и полу-чения списка связаны *только* через вызов функции **C_DecryptDigestUpdate**, эти 2 байта открытого текста *не* передаются для вычисления списка.

Следовательно, вызов **C_DigestFinal** вычислит список *первых 16 байт открытого текста*, а не список всего открытого текста. Решающим является то, что еще до вызова функции **C_DigestFinal** последние 2 байта открытого текста передаются активной операции вычисления списка через вызов **C_DigestUpdate.**

Поэтому важно, чтобы, когда приложение использует механизм расшифрования с заполнением при помощи функции **C_DecryptDigestUpdate**, оно точно знало, сколько открытого текста было передано активной операции вычисления списка. *Подчеркиваем необходимость соблюдения особой осторожности при использовании механизма расшифрования с заполнением при помощи функции **C_DecryptDigestUpdate**.*

Возвращаемые значения: 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.

Пример:

```
#define BUF_SZ 512

CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_BYTE iv[8];
CK_MECHANISM decryptionMechanism = {
  CKM_DES_ECB, iv, sizeof(iv)
};
CK_MECHANISM digestMechanism = {
  CKM_MD5, NULL_PTR, 0
};
CK_BYTE encryptedData[(2*BUF_SZ)+8];
CK_BYTE digest[16];
CK_ULONG ulDigestLen;
CK_BYTE data[BUF_SZ];
CK_ULONG ulDataLen, ulLastUpdateSize;
CK_RV rv;

.
.
memset(iv, 0, sizeof(iv));
memset(encryptedData, 'A', ((2*BUF_SZ)+8));
rv = C_DecryptInit(hSession, &decryptionMechanism, hKey);
if (rv != CKR_OK) {
  .
  .
}
rv = C_DigestInit(hSession, &digestMechanism);
if (rv != CKR_OK){
  .
  .
}
```

```
ulDataLen = sizeof(data);
rv = C_DecryptDigestUpdate(
  hSession,
  &encryptedData[0], BUF_SZ,
  data, &ulDataLen);
.
.
ulDataLen = sizeof(data);
rv = C_DecryptDigestUpdate(
  hSession,
  &encryptedData[BUF_SZ], BUF_SZ,
  data, &ulDataLen);
.
.

/*
 * The last portion of the buffer needs to be handled with
 * separate calls to deal with padding issues in ECB mode
 */

/* First, complete the decryption of the buffer */
ulLastUpdateSize = sizeof(data);
rv = C_DecryptUpdate(
  hSession,
  &encryptedData[BUF_SZ*2], 8,
  data, &ulLastUpdateSize);
.
.
/* Get last piece of plaintext (should have length 0, here) */
ulDataLen = sizeof(data)-ulLastUpdateSize;
rv = C_DecryptFinal(hSession, &data[ulLastUpdateSize], &ulDataLen);
if (rv != CKR_OK) {
  .
  .
}

/* Digest last bit of plaintext */
rv = C_DigestUpdate(hSession, &data[BUF_SZ*2], 5);
if (rv != CKR_OK) {
  .
  .
}
ulDigestLen = sizeof(digest);
rv = C_DigestFinal(hSession, digest, &ulDigestLen);
if (rv != CKR_OK) {
  .
  .
}
```

♦ **C_SignEncryptUpdate**

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

C_SignEncryptUpdate продолжает комбинированную многостороннюю операцию постановки электронной подписи и шифрования, обрабатывая еще одну часть данных. *hSession* – идентификатор сеанса; *pPart* указывает на часть данных; *ulPartLen* – длина элемента данных; *pEncryptedPart* указывает на адрес участка памяти, в который передается зашифрованная и прошедшая вычисление списка часть данных; *pulEncryptedPart* указывает на адрес участка памяти, который содержит информацию о длине зашифрованной части данных.

Производя вывод данных, **C_SignEncryptUpdate** придерживается норм, описанных в 11.2. Если вызов **C_SignEncryptUpdate** не выдает зашифрованных данных (что связано с ошибками, или с тем, что *pEncryptedPart* имеет значение NULL_PTR, или потому, что длина поля *pulEncryptedPartLen* недостаточна, чтобы содержать всю выводимую часть зашифрованных данных), то никакой открытый текст не передается активной операции электронной подписи.

Активными должны быть операции шифрования и электронной подписи (они должны быть инициализированы с использованием функций **C_SignInit** и **C_EncryptInit** соответственно). Данная функция может вызываться любое количество раз подряд и чередоваться с вызовами **C_SignUpdate** и **C_EncryptUpdate**.

Возвращаемые значения: 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.

Пример:

```
#define BUF_SZ 512

CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hEncryptionKey, hMacKey;
CK_BYTE iv[8];
CK_MECHANISM signMechanism = {
  CKM_DES_MAC, NULL_PTR, 0
};
CK_MECHANISM encryptionMechanism = {
  CKM_DES_ECB, iv, sizeof(iv)
};
CK_BYTE encryptedData[BUF_SZ];
CK_ULONG ulEncryptedDataLen;
CK_BYTE MAC[4];
CK_ULONG ulMacLen;
CK_BYTE data[(2*BUF_SZ)+8];
CK_RV rv;
int i;

.
.
memset(iv, 0, sizeof(iv));
memset(data, 'A', ((2*BUF_SZ)+5));
rv = C_EncryptInit(hSession, &encryptionMechanism, hEncryptionKey);
if (rv != CKR_OK) {
  .
  .
}



rv = C_SignInit(hSession, &signMechanism, hMacKey);
if (rv != CKR_OK) {
  .
  .
}

ulEncryptedDataLen = sizeof(encryptedData);
rv = C_SignEncryptUpdate(
  hSession,
  &data[0], BUF_SZ,
  encryptedData, &ulEncryptedDataLen);
.
.
ulEncryptedDataLen = sizeof(encryptedData);
rv = C_SignEncryptUpdate(
  hSession,
  &data[BUF_SZ], BUF_SZ,
```

```
  encryptedData, &ulEncryptedDataLen);
.
.

/*
 * The last portion of the buffer needs to be handled with
 * separate calls to deal with padding issues in ECB mode
 */

/* First, complete the signature on the buffer */
rv = C_SignUpdate(hSession, &data[BUF_SZ*2], 5);
.
.
ulMacLen = sizeof(MAC);
rv = C_SignFinal(hSession, MAC, &ulMacLen);
.
.

/* Then pad last part with 3 0x00 bytes, and complete encryption */
for(i=0;i<3;i++)
  data[((BUF_SZ*2)+5)+i] = 0x00;

/* Now, get second-to-last piece of ciphertext */
ulEncryptedDataLen = sizeof(encryptedData);
rv = C_EncryptUpdate(
  hSession,
  &data[BUF_SZ*2], 8,
  encryptedData, &ulEncryptedDataLen);
.
.

/* Get last piece of ciphertext (should have length 0, here) */
ulEncryptedDataLen = sizeof(encryptedData);
rv = C_EncryptFinal(hSession, encryptedData, &ulEncryptedDataLen);
.
.
```

♦ **C_DecryptVerifyUpdate**

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

C_DecryptVerifyUpdate продолжает многостороннюю объединенную операцию расшифрования и верификации электронной подписи, обрабатывая еще одну часть данных. *hSession* – идентификатор сеанса; *pEncryptedPart* указывает на зашифрованную часть данных; *ulEncryptedPartLen* – длина части зашифрованных данных; *pPart* указывает на адрес участка памяти, в который поступают восстановленные данные; *pulPartLen* указывает на адрес участка памяти, в котором содержится информация о длине восстановленных данных.

Производя вывод данных, **C_DecryptVerifyUpdate** придерживается норм, описание которых приводится в 11.2. Если вызов **C_DecryptVerifyUpdate** не выдает результирующего вывода расшифрованных данных (из-за ошибок, или того, что *pPart* имеет значение NULL_PTR, или потому, что размер поля *pulPartLen* недостаточен, чтобы содержать всю выдаваемую часть расшифрованных данных), то никакой открытый текст не передается активной операции верификации.

Активными должны быть операции расшифровки и электронной подписи (они должны быть инициализированы с использованием функций **C_DecryptInit** и **C_VerifyInit** соответственно). Данная функция может вызываться любое количество раз подряд и чередоваться с вызовами **C_DecryptUpdate** и **C_VerifyUpdate**.

При использовании функции **C_DecryptVerifyUpdate** появляется вопрос сцепления [конвейеризации], который не возникает при использовании **C_SignEncryptUpdate**, «обратной функции»

C_DecryptDigestUpdate. Это происходит потому, что при вызове **C_SignEncryptUpdate** абсолютно одинаковые данные ввода передаются как активной операции получения списка, так и активной операции зашифровывания; однако, когда вызывается **C_DecryptVerifyUpdate**, данные ввода, передаваемые активной операции получения списка, являются *данными вывода* активной операции расшифровки. Этот вопрос возникает только тогда, когда механизм, применяемый для расшифровки, использует заполнение.

Представим, в частности, 24-байтный шифртекст, который был получен посредством зашифровывания 18-байтного открытого текста алгоритмом DES в режиме CBC с заполнением PKCS. Рассмотрим приложение, которое будет одновременно расшифровывать этот шифртекст и выполнять верификацию подписи получаемого таким образом исходного открытого текста.

После инициализации операций расшифровки и верификации приложение передает 24-байтный шифртекст (3 блока DES) функции **C_DecryptVerifyUpdate**. **C_DecryptVerifyUpdate** возвращает ровно 16 байт открытого текста, так как в данный момент Cryptoki не знает, последует ли еще порция шифр-текста и содержит ли последний блок шифртекста какое-либо заполнение. Эти 16 байт открытого
текста передаются активной операции верификации.

Так как шифртекста больше нет, приложение вызывает **C_DecryptFinal**. Тем самым Cryptoki уведомляется, что новая порция шифртекста не последует, и результатом данного вызова становятся последние 2 байта открытого текста. Однако, поскольку активные операции расшифровки и верификации связаны *только* через вызов функции **C_DecryptVerifyUpdate**, эти 2 байта открытого текста *не* передаются для верификации.

Следовательно, вызов **C_VerifyFinal** проведет верификацию подписи на *первых 16 байтах открытого текста*, а не на всем открытом тексте. Решающим является то, что еще до вызова функции **C_VerifyFinal** последние 2 байта открытого текста передаются активной операции верификации через вызов **C_VerifyUpdate.**

Поэтому важно, чтобы, когда приложение использует механизм расшифровки с заполнением при помощи функции **C_DecryptVerifyUpdate**, оно точно знало, сколько открытого текста было передано активной операции вычисления списка. *Подчеркиваем необходимость соблюдения особой осторожности при использовании механизма расшифровки с заполнением при помощи функции **C_DecryptVerifyUpdate**.*

Возвращаемые значения: 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_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.

Пример:

```
#define BUF_SZ 512

CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hDecryptionKey, hMacKey;
CK_BYTE iv[8];
CK_MECHANISM decryptionMechanism = {
  CKM_DES_ECB, iv, sizeof(iv)
};
CK_MECHANISM verifyMechanism = {
  CKM_DES_MAC, NULL_PTR, 0
};
CK_BYTE encryptedData[(2*BUF_SZ)+8];
CK_BYTE MAC[4];
CK_ULONG ulMacLen;
CK_BYTE data[BUF_SZ];
CK_ULONG ulDataLen, ulLastUpdateSize;
CK_RV rv;

.
.
memset(iv, 0, sizeof(iv));
memset(encryptedData, 'A', ((2*BUF_SZ)+8));
rv = C_DecryptInit(hSession, &decryptionMechanism, hDecryptionKey);
if (rv != CKR_OK) {
```

```
  .
  .
}
rv = C_VerifyInit(hSession, &verifyMechanism, hMacKey);
if (rv != CKR_OK){
  .
  .
}

ulDataLen = sizeof(data);
rv = C_DecryptVerifyUpdate(
  hSession,
  &encryptedData[0], BUF_SZ,
  data, &ulDataLen);
.
.
ulDataLen = sizeof(data);
rv = C_DecryptVerifyUpdate(
  hSession,
  &encryptedData[BUF_SZ], BUF_SZ,
  data, &uldataLen);
.
.

/*
 * The last portion of the buffer needs to be handled with
 * separate calls to deal with padding issues in ECB mode
 */

/* First, complete the decryption of the buffer */
ulLastUpdateSize = sizeof(data);
rv = C_DecryptUpdate(
  hSession,
  &encryptedData[BUF_SZ*2], 8,
  data, &ulLastUpdateSize);
.
.
/* Get last little piece of plaintext. Should have length 0 */
ulDataLen = sizeof(data)-ulLastUpdateSize;
rv = C_DecryptFinal(hSession, &data[ulLastUpdateSize], &ulDataLen);
if (rv != CKR_OK) {
  .
  .
}

/* Send last bit of plaintext to verification operation */
rv = C_VerifyUpdate(hSession, &data[BUF_SZ*2], 5);
if (rv != CKR_OK) {
  .
  .
}
```

```
rv = C_VerifyFinal(hSession, MAC, ulMacLen);
if (rv == CKR_SIGNATURE_INVALID) {
  .
  .
}
```

11.14 Функции управления ключами

Cryptoki предоставляет следующие функции для управления ключами:

- **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 генерирует секретный ключ или набор параметров домена, создавая новый объект. *hSession* – идентификатор сеанса; *pMechanism* указывает на механизм генерации; *pTemplate* указывает на шаблон для нового ключа или набора параметров домена; *ulCount* – количество атрибутов в шаблоне; *phKey* указывает на адрес участка памяти, в которую передается идентификатор нового ключа или набор параметров домена.

Если механизм генерации предназначен для генерации параметров домена, атрибут **CKA_CLASS** будет иметь значение CKO_DOMAIN_PARAMETERS; если нет – он будет иметь значение CKO_ SECRET_KEY.

Поскольку тип ключа или параметров домена, которые будут генерироваться, определен в самом механизме генерации, шаблону не нужно предоставлять тип ключа. Если в нем все же указан тип ключа, противоречащий механизму генерации, функция **C_GenerateKey** завершится сбоем с кодом ошибки CKR_TEMPLATE_INCONSISTENT. Аналогично рассматривается атрибут CKA_CLASS.

Если вызов функции **C_GenerateKey** не может поддерживать именно тот шаблон, который ей предоставлен, данный вызов завершится сбоем и выдаст возвращаемое значение без создания объекта.

В объекте, созданном при удачном вызове функции **C_GenerateKey**, атрибут **CKA_LOCAL** будет выставлен на CK_TRUE.

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

Пример:

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

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

- **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 генерирует пару открытый/секретный ключ, создавая новые объекты ключей. *hSession* – идентификатор сеанса; *pMechanism* указывает на механизм генерации ключа; *pPublicKey-Template* указывает на шаблон для открытого ключа; *ulPublicKeyAttributeCount* – количество атрибутов в шаблоне открытого ключа; *pPrivateKeyTemplate* указывает на шаблон секретного ключа; *ulPrivateKey-AttributeCount* – количество атрибутов в шаблоне секретного ключа; *phPublicKey* указывает на адрес участка памяти, в которую поступает идентификатор нового личного ключа.

Поскольку типы ключей, которые будут генерироваться, определены в самом механизме генерации, шаблону не нужно предоставлять тип ключа. Если же в одном из шаблонов все же указан тип ключа, противоречащий механизму генерации, функция **C_GenerateKeyPair** завершится сбоем с кодом ошибки CKR_TEMPLATE_INCONSISTENT. Аналогично рассматривается атрибут CKA_CLASS.

Если вызов функции **C_GenerateKeyPair** не может поддерживать именно те шаблоны, которые ей предоставлены, данный вызов завершится сбоем и выдаст возвращаемое значение без создания объекта.

Вызов **C_GenerateKeyPair** никогда не создает только один ключ. Вызов может завершиться сбоем, не создав ключей, или он может завершиться успешно, создав согласованную пару открытый/ секретный ключ.

В объектах ключей, созданных при удачном завершении вызова функции **C_GenerateKeyPair**, атрибуты **CKA_LOCAL** будут выставлены на CK_TRUE.

*Внимательно следите за порядком аргументов в **C_GenerateKeyPair**. Порядок следования двух последних аргументов – не тот, который был у них в первоначальной версии Cryptoki 1.0. Порядок следования этих двух аргументов стал причиной некоторой нежелательной путаницы.*

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

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hPublicKey, hPrivateKey;
CK_MECHANISM mechanism = {
  CKM_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) {
  .
  .
}
```

♦ **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 упаковывает (т. е. шифрует) частный/секретный или секретный ключ. *hSession* – идентификатор сеанса; *pMechanism* указывает на механизм упаковки; *hWrappingKey* – идентификатор упаковочного ключа; *hKey* – идентификатор ключа, который будет упакован; *pWrappedKey* указывает на адрес участка памяти, в который направляется упакованный ключ; *pulWrappedKeyLen* указывает на ячейку памяти, в которую поступает информация о длине упакованного ключа.

Производя вывод данных, **C_WrapKey** придерживается норм, описанных в 11.2.

Атрибут **CKA_WRAP** упаковочного ключа, который указывает, поддерживает ли данный ключ упаковку, должен быть выставлен на CK_TRUE. Атрибут **CKA_EXTRACTABLE** ключа, который будет упакован, также должен быть выставлен на CK_TRUE.

Если ключ, который должен быть упакован, не может быть упакован по причинам, связанным с особенностями носителя, несмотря на то, что его атрибут **CKA_EXTRACTABLE** выставлен на CK_TRUE, то функция **C_WrapKey** завершится сбоем с кодом ошибки CKR_KEY_NOT_WRAPPABLE. Если же он не может быть упакован с использованием указанного упаковочного ключа или механизма исключительно по причине его длины, то **C_WrapKey** завершится сбоем с кодом ошибки CKR_KEY_SIZE_ RANGE.

C_WrapKey может использоваться в следующих ситуациях:

- для упаковки любого секретного ключа при помощи открытого ключа, поддерживающего шифрование и расшифрование;
- для упаковки любого секретного ключа с использованием любого другого секретного ключа. Необходимо принимать во внимание размер ключа и мощность механизма, или носитель может не позволить данную операцию;
- для упаковки частного/секретного ключа с использованием любого секретного ключа.

Конечно, носители отличаются по тому, какие типы ключей они могут упаковать и с использованием каких механизмов они это могут делать.

Для распределения упаковочных ключей так, чтобы они могли упаковывать только подмножество извлекаемых ключей, можно использовать атрибут CKA_WRAP_TEMPLATE упаковочного ключа, при помощи которого можно указать множество атрибутов, которые будут сопоставляться с атрибутами ключа, подлежащего упаковке. Если все атрибуты совпадают в соответствии с правилами сопоставления атрибутов из C_FindObject, то операция упаковки будет начата. Значение данного атрибута является шаблоном атрибутов, а размер – это количество элементов в шаблоне, умноженное на размер CK_ATTRIBUTE. Если данный атрибут не предоставлен, то приниматься может любой шаблон. Атрибуты, которые отсутствуют, не проверяются. Если при попытке упаковать ключ произойдет какое-либо несовпадение атрибутов, функция выдаст CKR_KEY_HANDLE_INVALID.

Возвращаемые значения: 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.

Пример:

```
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) {
  .
  .
}
```

♦ **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 распаковывает (т. е. расшифровывает) упакованный ключ, создавая новый объект частного/секретного или секретного ключа. *hSession* – идентификатор сеанса; *pMechanism* указывает на механизм распаковки; *hUnwrappingKey* – идентификатор механизма распаковки; *hUnwrappingKey* – идентификатор распаковочного ключа; *pWrappedKey* указывает на распакованный ключ; *ulWrapped-KeyLen* – длина распакованного ключа; *pTemplate* указывает на шаблон для нового ключа; *ulAttribute-Count* – количество атрибутов в шаблоне; *phKey* указывает на адрес участка памяти, в который поступает идентификатор извлеченного ключа.

Атрибут **CKA_UNWRAP** распаковочного ключа, который указывает, поддерживает ли данный ключ распаковку, должен быть выставлен на CK_TRUE.

В новом ключе атрибут **CKA_ALWAYS_SENSITIVE** будет выставлен на CK_FALSE и атрибут **CKA_NEVER_EXTRACTABLE** – на CK_FALSE. Атрибут **CKA_EXTRACTABLE** по умолчанию выставлен на CK_TRUE.

Некоторые механизмы могут изменять или пытаться изменять содержимое структуры pMechanism во время распаковки ключа.

Если вызов функции **C_UnwrapKey** не может поддерживать именно тот шаблон, который ей предоставлен, данный вызов завершится сбоем и выдаст возвращаемое значение без создания объекта ключа.

В объекте ключа, созданном при удачном завершении вызова функции **C_UnwrapKey**, атрибут **CKA_LOCAL** будет выставлен на CK_FALSE.

Для распределения распаковочных ключей так, чтобы они могли распаковывать только подмножество извлекаемых ключей, можно использовать атрибут CKA_WRAP_TEMPLATE распаковочного ключа, при помощи которого можно указать множество атрибутов, которые будут добавляться к атрибутам ключа, подвергающегося распаковке. Если эти атрибуты не конфликтуют с шаблоном атрибутов, заданным пользователем в 'pTemplate', то распаковка будет начата. Значение данного атрибута является шаблоном атрибутов, а размер – это количество элементов в шаблоне, умноженное на размер CK_ATTRIBUTE. Если данный атрибут отсутствует в распаковочном ключе, то дополнительные атрибуты добавляться не будут. Если при попытке распаковать ключ будет иметь место какой-либо
конфликт атрибутов, то функция выдаст CKR_TEMPLATE_INCONSISTENT.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_ TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_BUFFER_TOO_SMALL, 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_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.

Пример:

```
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) {
  .
  .
}
```

- **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 выводит ключ из базового ключа, создавая при этом новый объект ключа. *hSession* – идентификатор сеанса; *pMechanism* указывает на структуру, которая задает механизм извлечения ключа; *hBaseKey* – идентификатор базового ключа; *pTemplate* указывает на шаблон для нового ключа; *ulAttributeCount* – количество атрибутов в шаблоне; *phKey* указывает на адрес участка памяти, в который поступает извлеченный ключ.

Значения атрибутов **CK_SENSITIVE**, **CK_ALWAYS_SENSITIVE**, **CK_EXTRACTABLE** и **CK_NEVER_EXTRACTABLE** базового ключа влияют на значения, которые данные атрибуты могут содержать для нового выведенного ключа (см. описание каждого из механизмов вывода ключей в 11.17.2 для получения информации по каким-либо ограничениям такого рода).

В объекте ключа, созданном при удачном завершении вызова функции **C_DeriveKey**, атрибут **CKA_LOCAL** будет выставлен на CK_FALSE.

Если вызов функции **C_DeriveKey** не может поддерживать именно тот шаблон, который ей предоставлен, данный вызов завершится сбоем и выдаст возвращаемое значение без создания объекта ключа.

Возвращаемые значения: 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_I NVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_ INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_ NOT_LOGGED_IN.

Пример:

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hPublicKey, hPrivateKey, hKey;
CK_MECHANISM keyPairMechanism = {
  CKM_DH_PKCS_KEY_PAIR_GEN, NULL_PTR, 0
};
CK_BYTE prime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE publicValue[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) {
  .
  .
  }
  }
}
```

11.15 Функции генерации случайных чисел

Cryptoki предоставляет следующие функции для генерации случайных чисел:

- **C_SeedRandom**

```
CK_DEFINE_FUNCTION(CK_RV, C_SeedRandom)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pSeed,
  CK_ULONG ulSeedLen
);
```

C_SeedRandom добавляет дополнительный набор начальных чисел в генератор случайных чисел носителя. *hSession* – идентификатор сеанса; *pSeed* указывает на набор начальных чисел; *ulSeedLen* – длина набора начальных чисел в байтах.

Возвращаемые значения: 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.

Пример: см. C_GenerateRandom.

- **C_GenerateRandom**

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

C_GenerateRandom генерирует случайные или псевдослучайные данные. *hSession* – идентификатор сеанса; *pRandomData* указывает на адрес участка памяти, в который направляются случайные данные; *ulRandomLen* – длина в байтах случайных или псевдослучайных данных, которые должны быть сгенерированы.

Возвращаемые значения: CKR_ARGUMENTS_BAD, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_ FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_OK, CKR_OPERATION_ACTIVE, CKR_RANDOM_NO_RNG, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_ NOT_LOGGED_IN.

Пример:

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

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

11.16 Функции управления параллельными функциями

Cryptoki предоставляет следующие функции для управления параллельным выполнением криптографических функций. Данные функции существуют только для обратной совместимости.

- **C_GetFunctionStatus**

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

В предыдущих версиях Cryptoki **C_GetFunctionStatus** использовалась для получения информации о состоянии функции, выполняемой параллельно с приложением. Однако в данной версии **C_GetFunctionStatus** – это унаследованная функция, которая должна просто выдавать значение CKR_FUNCTION_NOT_PARALLEL.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_FUNCTION_ NOT_PARALLEL, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_CLOSED.

- **C_CancelFunction**

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

В предыдущих версиях Cryptoki **C_CancelFunction** отменяла функцию, выполняемую параллельно с приложением. Однако в данной версии **C_CancelFunction** – унаследованная функция, которая должна просто выдавать значение CKR_FUNCTION_NOT_PARALLEL.

Возвращаемые значения: CKR_CRYPTOKI_NOT_INITIALIZED, CKR_FUNCTION_FAILED, CKR_FUNCTION_ NOT_PARALLEL, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_CLOSED.

11.17 Функции обратных вызовов

Сеансы Cryptoki могут использовать указатели функций типа **CK_NOTIFY**, чтобы уведомлять приложение о каких-либо событиях.

11.17.1 Обратные вызовы передачи управления

Криптографические функции (т. е. любые функции, подходящие под одну из следующих категорий: функции шифрования, функции расшифрования, функции получения списка сообщения, функции электронной подписи и MAC, функции верификации электронной подписи и MAC, криптографические функции двойного назначения, функции управления ключами, функции генерации случайных чисел), которые выполняются в сеансах Cryptoki, могут периодически передавать управление приложению, которое вызывает их, если сеанс, в котором они выполняются, при его открытии имел связанную с этим уведомительную функцию обратного вызова. Функции осуществляют это, обращаясь к обратному вызову сеанса с использованием аргументов (`hSession, CKN_SURRENDER, pApplication`), где `hSession` – идентификатор сеанса, а pApplication предоставлен функции **C_OpenSession** при открытии сеанса. Обратные вызовы передачи управления должны выдавать значение CKR_OK (которое указывает, что Cryptoki будет продолжать выполнять данную функцию) или значение CKR_CANCEL (которое указывает, что Cryptoki прервет выполнение данной функции). Перед тем как возвратить одно из таких значений, обратный вызов функции выполнит некоторые расчеты, если это необходимо.

Типичным примером использования обратного вызова передачи управления может быть обратная связь приложения с пользователем во время операции генерации длинной пары ключей. Каждый раз, когда приложение получает обратный вызов, оно должно отображать для пользователя дополнительный символ «.». Оно также может проверить действия в буфере клавиатуры с момента последнего обратного вызова передачи управления и прервать операцию генерации пары ключей (возможно, возвратив значение CKR_CANCEL), если пользователь нажал <ESCAPE>.

Библиотека Cryptoki *не обязательно должна* делать *какие-либо* обратные вызовы передачи управления.

11.17.2 Обратные вызовы, определенные поставщиком

Поставщики библиотек также могут определять дополнительные типы обратных вызовов. В связи с наличием данной возможности расширения подпрограммы приложения, работающие с обратными вызовами уведомлений, должны проверять все обратные вызовы, которые они получают, и, если такой тип обратного вызова им не знаком, должны немедленно передать управление обратно библиотеке, возвратив при этом значение CKR_OK.

12 Механизмы

Любой механизм точно определяет, как должен осуществляться определенный криптографический процесс.

В таблице 34 показано, какие механизмы из Cryptoki поддерживаются различными криптографическими операциями. Для любого конкретного носителя какая-то из операций может удовлетворительно поддерживать лишь определенный набор из перечисленных механизмов. Кроме того, нет гарантии, что один носитель, поддерживающий один механизм для какой-либо операции, поддерживает и любой другой механизм для любой другой операции (или даже поддерживает тот же механизм для любой другой операции). Например, если носитель может создавать цифровые подписи стандарта RSA посредством механизма **CKM_RSA_PKCS**, этот носитель может как осуществлять, так и не осуществлять шифрование RSA посредством **CKM_RSA_PKCS**.

Таблица 34 – Соответствие механизмов и функций

Механизм	Функции						
	Зашифровать и расшифровать	Подписать и удостоверить	SR и VR [1]	Список	Генерация ключа/пары ключей	Упаковка и распаковка	Деривация
CKM_RSA_PKCS_KEY_PAIR_GEN					✓		
CKM_RSA_X9_31_KEY_PAIR_GEN					✓		
CKM_RSA_PKCS	✓[2]	✓[2]	✓			✓	
CKM_RSA_PKCS_OAEP	✓[2]					✓	
CKM_RSA_PKCS_PSS		✓[2]					
CKM_RSA_9796		✓[2]	✓				
CKM_RSA_X_509	✓[2]	✓[2]	✓			✓	
CKM_RSA_X9_31		✓[2]					
CKM_MD2_RSA_PKCS		✓					
CKM_MD5_RSA_PKCS		✓					
CKM_SHA1_RSA_PKCS		✓					
CKM_SHA256_RSA_PKCS		✓					
CKM_SHA384_RSA_PKCS		✓					
CKM_SHA512_RSA_PKCS		✓					
CKM_RIPEMD128_RSA_PKCS		✓					
CKM_RIPEMD160_RSA_PKCS		✓					
CKM_SHA1_RSA_PKCS_PSS		✓					
CKM_SHA256_RSA_PKCS_PSS		✓					
CKM_SHA384_RSA_PKCS_PSS		✓					

Продолжение таблицы 34

Механизм	Функции						
	Зашифровать и расшифровать	Подписать и удостоверить	SR и VR [1]	Список	Генерация ключа/пары ключей	Упаковка и распаковка	Деривация
CKM_SHA512_RSA_PKCS_PSS		✓					
CKM_SHA1_RSA_X9_31		✓					
CKM_DSA_KEY_PAIR_GEN					✓		
CKM_DSA_PARAMETER_GEN					✓		
CKM_DSA		✓[2]					
CKM_DSA_SHA1		✓					
CKM_FORTEZZA_TIMESTAMP		✓[2]					
CKM_EC_KEY_PAIR_GEN (CKM_ECDSA_KEY_PAIR_GEN)					✓		
CKM_ECDSA		✓[2]					
CKM_ECDSA_SHA1		✓					
CKM_ECDH1_DERIVE							✓
CKM_ECDH1_COFACTOR_DERIVE							✓
CKM_ECMQV_DERIVE							✓
CKM_DH_PKCS_KEY_PAIR_GEN					✓		
CKM_DH_PKCS_PARAMETER_GEN					✓		
CKM_DH_PKCS_DERIVE							✓
CKM_X9_42_DH_KEY_PAIR_GEN					✓		
CKM_X9_42_DH_PKCS_PARAMETER_GEN					✓		
CKM_X9_42_DH_DERIVE							✓
CKM_X9_42_DH_HYBRID_DERIVE							✓
CKM_X9_42_MQV_DERIVE							✓
CKM_KEA_KEY_PAIR_GEN					✓		
CKM_KEA_KEY_DERIVE							✓
CKM_GENERIC_SECRET_KEY_GEN					✓		
CKM_RC2_KEY_GEN					✓		
CKM_RC2_ECB	✓					✓	
CKM_RC2_CBC	✓					✓	
CKM_RC2_CBC_PAD	✓					✓	
CKM_RC2_MAC_GENERAL		✓					
CKM_RC2_MAC		✓					
CKM_RC4_KEY_GEN					✓		
CKM_RC4	✓						
CKM_RC5_KEY_GEN					✓		
CKM_RC5_ECB	✓					✓	
CKM_RC5_CBC	✓					✓	
CKM_RC5_CBC_PAD	✓					✓	
CKM_RC5_MAC_GENERAL		✓					
CKM_RC5_MAC		✓					
CKM_AES_KEY_GEN					✓		
CKM_AES_ECB	✓					✓	
CKM_AES_CBC	✓					✓	
CKM_AES_CBC_PAD	✓					✓	
CKM_AES_MAC_GENERAL		✓					

Продолжение таблицы 34

Механизм	Функции						
	Зашифровать и расшифровать	Подписать и удостоверить	SR и VR [1]	Список	Генерация ключа/пары ключей	Упаковка и распаковка	Деривация
CKM_AES_MAC		✓					
CKM_DES_KEY_GEN					✓		
CKM_DES_ECB	✓					✓	
CKM_DES_CBC	✓					✓	
CKM_DES_CBC_PAD	✓					✓	
CKM_DES_MAC_GENERAL		✓					
CKM_DES_MAC		✓					
CKM_DES2_KEY_GEN					✓		
CKM_DES3_KEY_GEN					✓		
CKM_DES3_ECB	✓					✓	
CKM_DES3_CBC	✓					✓	
CKM_DES3_CBC_PAD	✓					✓	
CKM_DES3_MAC_GENERAL		✓					
CKM_DES3_MAC		✓					
CKM_CAST_KEY_GEN					✓		
CKM_CAST_ECB	✓					✓	
CKM_CAST_CBC	✓					✓	
CKM_CAST_CBC_PAD	✓					✓	
CKM_CAST_MAC_GENERAL		✓					
CKM_CAST_MAC		✓					
CKM_CAST3_KEY_GEN					✓		
CKM_CAST3_ECB	✓					✓	
CKM_CAST3_CBC	✓					✓	
CKM_CAST3_CBC_PAD	✓					✓	
CKM_CAST3_MAC_GENERAL		✓					
CKM_CAST3_MAC		✓					
CKM_CAST128_KEY_GEN (CKM_CAST5_KEY_GEN)					✓		
CKM_CAST128_ECB (CKM_CAST5_ECB)	✓					✓	
CKM_CAST128_CBC (CKM_CAST5_CBC)	✓					✓	
CKM_CAST128_CBC_PAD (CKM_CAST5_CBC_PAD)	✓					✓	
CKM_CAST128_MAC_GENERAL (CKM_CAST5_MAC_GENERAL)		✓					
CKM_CAST128_MAC (CKM_CAST5_MAC)		✓					
CKM_IDEA_KEY_GEN					✓		
CKM_IDEA_ECB	✓					✓	
CKM_IDEA_CBC	✓					✓	
CKM_IDEA_CBC_PAD	✓					✓	
CKM_IDEA_MAC_GENERAL		✓					
CKM_IDEA_MAC		✓					
CKM_CDMF_KEY_GEN					✓		

Продолжение таблицы 34

Механизм	Функции

	Зашифровать и расшифровать	Подписать и удостоверить	SR и VR [1]	Список	Генерация ключа/пары ключей	Упаковка и распаковка	Деривация
CKM_CDMF_ECB	✓					✓	
CKM_CDMF_CBC	✓					✓	
CKM_CDMF_CBC_PAD	✓					✓	
CKM_CDMF_MAC_GENERAL		✓					
CKM_CDMF_MAC		✓					
CKM_DES_ECB_ENCRYPT_DATA							✓
CKM_DES_CBC_ENCRYPT_DATA							✓
CKM_DES3_ECB_ENCRYPT_DATA							✓
CKM_DES3_CBC_ENCRYPT_DATA							✓
CKM_AES_ECB_ENCRYPT_DATA							✓
CKM_AES_CBC_ENCRYPT_DATA							✓
CKM_SKIPJACK_KEY_GEN					✓		
CKM_SKIPJACK_ECB64	✓						
CKM_SKIPJACK_CBC64	✓						
CKM_SKIPJACK_OFB64	✓						
CKM_SKIPJACK_CFB64	✓						
CKM_SKIPJACK_CFB32	✓						
CKM_SKIPJACK_CFB16	✓						
CKM_SKIPJACK_CFB8	✓						
CKM_SKIPJACK_WRAP						✓	
CKM_SKIPJACK_PRIVATE_WRAP						✓	
CKM_SKIPJACK_RELAYX						✓[3]	
CKM_BATON_KEY_GEN					✓		
CKM_BATON_ECB128	✓						
CKM_BATON_ECB96	✓						
CKM_BATON_CBC128	✓						
CKM_BATON_COUNTER	✓						
CKM_BATON_SHUFFLE	✓						
CKM_BATON_WRAP						✓	
CKM_JUNIPER_KEY_GEN					✓		
CKM_JUNIPER_ECB128	✓						
CKM_JUNIPER_CBC128	✓						
CKM_JUNIPER_COUNTER	✓						
CKM_JUNIPER_SHUFFLE	✓						
CKM_JUNIPER_WRAP						✓	
CKM_MD2				✓			
CKM_MD2_HMAC_GENERAL		✓					
CKM_MD2_HMAC		✓					
CKM_MD2_KEY_DERIVATION							✓
CKM_MD5				✓			
CKM_MD5_HMAC_GENERAL		✓					
CKM_MD5_HMAC		✓					
CKM_MD5_KEY_DERIVATION							✓
CKM_SHA_1				✓			

Продолжение таблицы 34

Механизм	Функции						
	Зашифровать и расшифровать	Подписать и удостоверить	SR и VR [1]	Список	Генерация ключа/пары ключей	Упаковка и распаковка	Деривация
CKM_SHA_1_HMAC_GENERAL		✓					
CKM_SHA_1_HMAC		✓					
CKM_SHA1_KEY_DERIVATION							✓
CKM_SHA256				✓			
CKM_SHA256_HMAC_GENERAL		✓					
CKM_SHA256_HMAC		✓					
CKM_SHA256_KEY_DERIVATION							✓
CKM_SHA384				✓			
CKM_SHA384_HMAC_GENERAL		✓					
CKM_SHA384_HMAC		✓					
CKM_SHA384_KEY_DERIVATION							✓
CKM_SHA512				✓			
CKM_SHA512_HMAC_GENERAL		✓					
CKM_SHA512_HMAC		✓					
CKM_SHA512_KEY_DERIVATION							✓
CKM_RIPEMD128				✓			
CKM_RIPEMD128_HMAC_GENERAL		✓					
CKM_RIPEMD128_HMAC		✓					
CKM_RIPEMD160				✓			
CKM_RIPEMD160_HMAC_GENERAL		✓					
CKM_RIPEMD160_HMAC		✓					
CKM_FASTHASH				✓			
CKM_PBE_MD2_DES_CBC					✓		
CKM_PBE_MD5_DES_CBC					✓		
CKM_PBE_MD5_CAST_CBC					✓		
CKM_PBE_MD5_CAST3_CBC					✓		
CKM_PBE_MD5_CAST128_CBC (CKM_PBE_MD5_CAST5_CBC)					✓		
CKM_PBE_SHA1_CAST128_CBC (CKM_PBE_SHA1_CAST5_CBC)					✓		
CKM_PBE_SHA1_RC4_128					✓		
CKM_PBE_SHA1_RC4_40					✓		
CKM_PBE_SHA1_DES3_EDE_CBC					✓		
CKM_PBE_SHA1_DES2_EDE_CBC					✓		
CKM_PBE_SHA1_RC2_128_CBC					✓		
CKM_PBE_SHA1_RC2_40_CBC					✓		
CKM_PBA_SHA1_WITH_SHA1_HMAC					✓		
CKM_PKCS5_PBKD2					✓		
CKM_KEY_WRAP_SET_OAEP						✓	
CKM_KEY_WRAP_LYNKS						✓	
CKM_SSL3_PRE_MASTER_KEY_GEN					✓		
CKM_SSL3_MASTER_KEY_DERIVE							✓
CKM_SSL3_MASTER_KEY_DERIVE_DH							✓
CKM_SSL3_KEY_AND_MAC_DERIVE							✓
CKM_SSL3_MD5_MAC		✓					

Окончание таблицы 34

Механизм	Функции						
	Зашифровать и расшифровать	Подписать и удостоверить	SR и VR [1]	Список	Генерация ключа/ пары ключей	Упаковка и распаковка	Деривация
CKM_SSL3_SHA1_MAC		✓					
CKM_TLS_PRE_MASTER_KEY_GEN					✓		
CKM_TLS_MASTER_KEY_DERIVE							✓
CKM_TLS_MASTER_KEY_DERIVE_DH							✓
CKM_TLS_KEY_AND_MAC_DERIVE							✓
CKM_TLS_PRF							✓
CKM_WTLS_PRE_MASTER_KEY_GEN					✓		
CKM_WTLS_MASTER_KEY_DERIVE							✓
CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC							✓
CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE							✓
CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE							✓
CKM_WTLS_PRF							✓
CKM_CMS_SIG		✓	✓				
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							✓

[1] SR = SignRecover (восстановление подписи – операция с подписью, при которой данные могут быть восстановлены из подписи), VR = VerifyRecover (верификационная операция, при которой данные восстанавливаются из подписи).
[2] Только односторонние операции (Single-part operations).
[3] Механизм может использоваться только для упаковки, но не для распаковки.

В остальной части данного раздела представлено подробное описание механизмов, поддерживаемых Criptoki, и передаваемых им параметров.

В общем, если механизм не ссылается на поля *ulMinKeyLen* и *ulMaxKeyLen* структуры CK_MECHANISM_INFO, то эти поля не используются этим конкретным механизмом.

12.1 RSA

12.1.1 Определения

В настоящем разделе определяется тип «CKK_RSA» ключа RSA для типа CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов RSA.

Механизмы:

```
CKM_RSA_PKCS_KEY_PAIR_GEN
CKM_RSA_PKCS
CKM_RSA_9796
CKM_RSA_X_509
CKM_MD2_RSA_PKCS
CKM_MD5_RSA_PKCS
CKM_SHA1_RSA_PKCS
CKM_SHA256_RSA_PKCS
CKM_SHA384_RSA_PKCS
CKM_SHA512_RSA_PKCS
CKM_RIPEMD128_RSA_PKCS
CKM_RIPEMD160_RSA_PKCS
```

```
CKM_RSA_PKCS_OAEP
CKM_RSA_X9_31_KEY_PAIR_GEN
CKM_RSA_X9_31
CKM_SHA1_RSA_X9_31
CKM_RSA_PKCS_PSS
CKM_SHA1_RSA_PKCS_PSS
CKM_SHA256_RSA_PKCS_PSS
CKM_SHA512_RSA_PKCS_PSS
CKM_SHA384_RSA_PKCS_PSS
```

12.1.2 Объекты открытого ключа RSA

Объекты открытого ключа RSA (класс объекта – **CKO_PUBLIC_KEY,** тип ключа – **CKK_RSA**) содержат открытые ключи RSA. В таблице 35 определены атрибуты объекта открытого ключа RSA в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 35 – Атрибуты объекта открытого ключа RSA

Атрибут	Тип данных	Значение
CKA_MODULUS [1, 4]	Большое целое	Модуль *n*
CKA_MODULUS_BITS [2, 3]	CK_ULONG	Длина в битах модуля *n*
CKA_PUBLIC_EXPONENT [1]	Большое целое	Открытый эспонент (Public exponent) *e*
См. сноски в таблице 15.		

В зависимости от носителя могут быть ограничения на длину компонентов ключа (см. PKCS #1 для дополнительной информации по ключам RSA).

Ниже представлен пример шаблона для создания объекта открытого ключа RSA:

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_RSA;
CK_UTF8CHAR label[] = "Объект открытого ключа RSA";
CK_BYTE modulus[] = {...};
CK_BYTE exponent[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
   {CKA_CLASS, &class, sizeof(class)},
   {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
   {CKA_TOKEN, &true, sizeof(true)},
   {CKA_LABEL, label, sizeof(label)-1},
   {CKA_WRAP, &true, sizeof(true)},
   {CKA_ENCRYPT, &true, sizeof(true)},
   {CKA_MODULUS, modulus, sizeof(modulus)},
   {CKA_PUBLIC_EXPONENT, exponent, sizeof(exponent)}
};
```

12.1.3 Объекты секретного ключа RSA

Объекты секретного ключа RSA (класс объекта – **CKO_ PRIVATE_KEY,** тип ключа – **CKK_RSA**) содержат секретные ключи RSA. В таблице 36 определены атрибуты объекта секретного ключа RSA в дополнение к общим атрибутам, определенным для данного класса объектов:

Таблица 36 – Атрибуты секретного ключа RSA

Атрибут	Тип данных	Значение
CKA_MODULUS [1, 4, 6]	Большое целое	Модуль *n*
CKA_PUBLIC_EXPONENT [4, 6]	Большое целое	Открытый экспонент *e*
CKA_PRIVATE_EXPONENT [1, 4, 6, 7]	Большое целое	Секретный экспонент *d*
CKA_PRIME_1 [4, 6, 7]	Большое целое	Простое число *p*
CKA_PRIME_2 [4, 6, 7]	Большое целое	Простое число *q*
CKA_EXPONENT_1 [4, 6, 7]	Большое целое	Секретный экспонент *d* по модулю *p*-1
CKA_EXPONENT_2 [4, 6, 7]	Большое целое	Секретный экспонент *d* по модулю *q*-1
CKA_COEFFICIENT [4, 6, 7]	Большое целое	Коэффициент CRT q^{-1} mod *p*

См. сноски в таблице 15.

В зависимости от носителя могут существовать ограничения на длину компонентов ключа (для дополнительной информации по ключам RSA см. документ PKCS #1).

Носители различаются тем, что именно они в действительности хранят для секретных ключей RSA. Некоторые носители хранят все указанные выше атрибуты, что может помочь в выполнении быстрых вычислений RSA. Другие носители могли бы хранить только значения **CKA_MODULUS** и **CKA_PRIVATE_EXPONENT**.

Из-за изложенного система Cryptoki проявляет гибкость при действиях с объектами секретных ключей RSA. Когда носитель генерирует секретный ключ RSA, он помещается для хранения в любой из отслеживаемых им участков полей, обозначенных в таблице 36. Затем, если приложение запросит значения различных атрибутов ключа, система Cryptoki выдаст значения только для тех атрибутов, значения которых она может получить (т. е. если систему Cryptoki запрашивают о значении атрибута, который она не может получить, то такой запрос не удовлетворяется). Следует отметить, что какая-либо конкретная реализация Cryptoki имеет или не имеет возможность и/или желание выдавать различные атрибуты секретных ключей RSA, которые в тот момент не хранятся на носителе. *Например*, если какой-то конкретный носитель хранит значения только для атрибутов **CKA_PRIVATE_ EXPONENT**, **CKA_PRIME_1** и **CKA_PRIME_2**, тогда Cryptoki *может* выдать значение для всех вышеупомянутых атрибутов (поскольку они могут быть успешно высчислены из этих трех значений). Однако какая-либо конкретная Cryptoki реально может или не может выполнять эти дополнительные вычисления. Лишь для атрибутов **CKA_MODULUS** и **CKA_PRIVATE_EXPONENT** из числа указанных в таблице 36 система Cryptoki *обязана* быть в состоянии выдать значения.

Если на носителе создается объект секретного ключа RSA и в ответ на запрос о создании объекта выдается больше атрибутов из числа указанных в таблице 36, чем может быть поддержано данным носителем, избыточные атрибуты, скорее всего, будут отбрасываться. Если делается попытка создать объект секретного ключа RSA с атрибутами, несущественными для именно этого носителя, тогда данный запрос о создании объекта удовлетворен не будет, а будет выдано сообщение CKR_TEMPLATE_INCOMPLETE.

Следует отметить, что при генерации секретного ключа RSA не задается атрибут **CKA_MODULUS_BITS**. Это происходит потому, что секретные ключи RSA генерируются лишь как части *пар* ключей RSA, а атрибут **CKA_MODULUS_BITS** для пары задается в шаблоне для открытого ключа RSA.

Ниже приводится образец шаблона для создания объекта секретного ключа RSA:

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_RSA;
CK_UTF8CHAR label[] = "Объект секретного ключа RSA";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE modulus[] = {...};
CK_BYTE publicExponent[] = {...};
CK_BYTE privateExponent[] = {...};
CK_BYTE prime1[] = {...};
CK_BYTE prime2[] = {...};
CK_BYTE exponent1[] = {...};
CK_BYTE exponent2[] = {...};
CK_BYTE coefficient[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {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_MODULUS, modulus, sizeof(modulus)},
```

```
  {CKA_PUBLIC_EXPONENT, publicExponent, sizeof(publicExponent)},
  {CKA_PRIVATE_EXPONENT, privateExponent, sizeof(privateExponent)},
  {CKA_PRIME_1, prime1, sizeof(prime1)},
  {CKA_PRIME_2, prime2, sizeof(prime2)},
  {CKA_EXPONENT_1, exponent1, sizeof(exponent1)},
  {CKA_EXPONENT_2, exponent2, sizeof(exponent2)},
  {CKA_COEFFICIENT, coefficient, sizeof(coefficient)}
};
```

12.1.4 Генерация пары ключей PKCS #1 RSA

Механизм генерации пар ключей PKCS #1 RSA, обозначенный как **CKM_RSA_PKCS_KEY_PAIR_GEN**, представляет собой механизм, основанный на криптосистеме RSA с открытым ключом, описанный в PKCS #1.

Данный механизм не имеет параметра.

Указанный механизм генерирует пары, состоящие из открытого и секретного ключей RSA с модулями определенной длины в битах и открытым экспонентом, как определено в атрибутах шаблона для открытого ключа **CKA_MODULUS_BITS** и **CKA_PUBLIC_EXPONENT**. Атрибут **CKA_PUBLIC_EXPONENT** может опускаться, тогда данный механизм будет выдавать открытый атрибут, используя значение по умолчанию 0x10001 (65537). Определенные реализации механизма могут использовать случайное значение либо альтернативное значение по умолчанию, если значение 0x10001 не сможет быть использовано данным носителем.

Примечание – Реализации данного механизма, строго соответствующие версии 2.11 или более ранним версиям, могут генерировать ошибку, если этот атрибут в шаблоне пропущен. Опыт показал, что многие реализации версии 2.11 и более ранних версий позволяли опускать в шаблоне атрибут CKA_PUBLIC_EXPONENT и вели себя, как описано выше. Данный механизм добавляет атрибуты CKA_CLASS, CKA_KEY_TYPE, CKA_MODULUS и CKA_PUBLIC_EXPONENT к новому открытому ключу. Атрибут CKA_PUBLIC_EXPONENT, если он будет выдан, скопируется с шаблона. Если приложение не сможет использовать выданное значение экспонента, будет выдано сообщение CKR_TEMPLATE_INCONSISTENT. Механизм добавляет атрибуты CKA_CLASS и CKA_KEY_TYPE к новому секретному ключу; он может также добавлять к новому секретному ключу некоторые из следующих атрибутов: CKA_MODULUS, CKA_PUBLIC_EXPONENT, CKA_PRIVATE_EXPONENT, CKA_PRIME_1, CKA_PRIME_2, CKA_EXPONENT_1, CKA_EXPONENT_2, CKA_COEFFICIENT. Другие атрибуты, поддерживаемые открытым и секретным типами ключей RSA (в частности, флаги, указывающие, какие функции поддерживает данный ключ), могут также быть заданы в шаблонах ключа, либо им могут быть приданы начальные значения по умолчанию.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* в структуре **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.1.5 Генерация пар ключей X9.31 RSA

Механизм генерации пар ключей X9.31 RSA, обозначенный как **CKM_RSA_X9_31_KEY_PAIR_GEN**, представляет собой механизм, основанный на криптосистеме RSA с открытым ключом, описанный в X9.31.

Данный механизм не имеет параметра.

Указанный механизм генерирует пары, состоящие из открытого и секретного ключей RSA с определенными модулями длины в битах и открытым экспонентом, как определено в атрибутах шаблона для открытого ключа **CKA_MODULUS_BITS** и **CKA_PUBLIC_EXPONENT**.

Этот механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_MODULUS** и **CKA_PUBLIC_EXPONENT** к новому открытому ключу. Механизм добавляет атрибуты **CKA_CLASS** и **CKA_KEY_TYPE** к новому секретному ключу; он может также добавлять к новому секретному ключу некоторые из следующих атрибутов: **CKA_MODULUS**, **CKA_PUBLIC_EXPONENT**, **CKA_PRIVATE_EXPONENT**, **CKA_PRIME_1**, **CKA_PRIME_2**, **CKA_EXPONENT_1**, **CKA_EXPONENT_2**, **CKA_COEFFICIENT**. Другие атрибуты, поддерживаемые различными типами открытых и секретных ключей RSA, могут также быть определены в шаблонах для ключей или же могут быть снабжены первичными значениями по умолчанию. В отличие от механизма **CKM_RSA_PKCS_KEY_PAIR_GEN** данный механизм гарантированно генерирует значения *p* и *q* для **CKA_PRIME_1** и **CKA_PRIME_2** соответственно, что отвечает строгому первичному требованию документа X9.31.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* в структуре **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.1.6 PKCS #1 v1.5 RSA

Механизм PKCS #1 v1.5 RSA, обозначенный как **CKM_RSA_PKCS**, представляет собой многоцелевой механизм, основанный на криптосистеме RSA с открытым ключом и форматами блоков, первично описанными в PKCS #1 v1.5. Данный механизм поддерживает одностороннее шифрование

и дешифрование; одностороннюю постановку электронной подписи и верификацию с восстановлением сообщения и без него, упаковку ключа и распаковку ключа. Этот механизм соответствует только той части документа PKCS #1 v1.5, которая включает RSA; он не вычисляет список сообщения и не осуществляет кодирование DigestInfo, как это определено для алгоритмов `md2withRSAEncryption` и `md5withRSAEncryption` в PKCS #1 v1.5.

Данный механизм не имеет параметра.

Указанный механизм может упаковывать и распаковывать любой секретный ключ соответствующей длины. Конкретный носитель может и не поддерживать упаковку/распаковку каждого поддерживаемого им секретного ключа соответствующей длины. Для упаковки «вводом» в операцию по дешифрованию является значение атрибута **CKA_VALUE** ключа, который упаковывается; аналогично и для распаковки. Этот механизм не упаковывает тип ключа или любую иную информацию о ключе, за исключением длины ключа; приложение должно производить это отдельно. В частности, в ходе распаковки данный механизм добавляет в восстанавливаемый ключ только атрибуты **CKA_CLASS** и **CKA_VALUE** (и **CKA_VALUE_LEN**, если данный ключ его имеет); другие атрибуты должны быть определены в шаблоне.

Ограничения на типы ключей и длину данных приведены в таблице 37. Для шифрования, дешифрования постановки подписи и верификации подписи ввод и вывод данных могут начинаться в одном и том же месте памяти. В таблице 37 k – длина модуля RSA в битах.

Таблица 37 – PKCS #1 v1.5 RSA: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt [1]	Открытый ключ RSA	$\leq k$-11	k	Тип блока 02
C_Decrypt [1]	Секретный ключ RSA	k	$\leq k$-11	Тип блока 02
C_Sign [1]	Секретный ключ RSA	$\leq k$-11	k	Тип блока 01
C_SignRecover	Секретный ключ RSA	$\leq k$-11	k	Тип блока 01
C_Verify[1]	Открытый ключ RSA	$\leq k$-11, k [2]	Не определена	Тип блока 01
C_VerifyRecover	Открытый ключ RSA	k	$\leq k$-11	Тип блока 01
C_WrapKey	Открытый ключ RSA	$\leq k$-11	k	Тип блока 02
C_UnwrapKey	Секретный ключ RSA	k	$\leq k$-11	Тип блока 02

[1] Только односторонние операции.
[2] Длина данных, длина подписи.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* в структуре **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.1.7 Параметры механизма OAEP для PKCS #1 RSA

♦ **CK_RSA_PKCS_MGF_TYPE; CK_RSA_PKCS_MGF_TYPE_PTR**

CK_RSA_PKCS_MGF_TYPE используется для отображения функции генерации сообщения (Message Generation Function, MGF), применяемой к блоку сообщения при форматировании блока сообщения по схеме OAEP (Optimal Asymmetric Encryption Padding – оптимальное асимметричное заполнение при зашифровывании) либо по схеме формирования цифровой подписи PKCS #1 PSS (Probabilistic Signature Scheme – вероятностная схема подписи). Данный механизм определяется следующим образом:

```
typedef CK_ULONG CK_RSA_PKCS_MGF_TYPE;
```

В PKCS #1 указаны следующие MGF. В таблице 38 перечислены указанные функции.

Таблица 38 – PKCS #1: Функции генерации маски

Идентификатор источника	Значение
CKG_MGF1_SHA1	0x00000001
CKG_MGF1_SHA256	0x00000002
CKG_MGF1_SHA384	0x00000003
CKG_MGF1_SHA512	0x00000004

CK_RSA_PKCS_MGF_TYPE_PTR является указателем на **CK_RSA_PKCS_ MGF_TYPE**.

♦ **CK_RSA_PKCS_OAEP_SOURCE_TYPE; CK_RSA_PKCS_OAEP_SOURCE_TYPE_PTR**

CK_RSA_PKCS_OAEP_SOURCE_TYPE применяется для указания источника параметра зашифровывания при форматировании блока сообщения для схемы зашифровывания PKCS #1 OAEP. Он определяется следующим образом:

```
typedef CK_ULONG CK_RSA_PKCS_OAEP_SOURCE_TYPE;
```

В PKCS #1 указаны следующие источники параметров зашифровывания. В таблице 39 перечислены указанные источники вместе с соответствующими типами данных для поля *pSourceData* в описанной ниже структуре **CK_RSA_PKCS_OAEP_PARAMS**.

Таблица 39 – PKCS #1 RSA OAEP: Источники параметра зашифровывания

Идентификатор источника	Значение	Тип данных
CKZ_DATA_SPECIFIED	0x00000001	Множество CK_BYTE, содержащее значение параметра зашифровывания. Если данный параметр является пустым, поле *pSourceData* должно быть NULL, а поле *ulSourceDataLen* должно быть нулем

CK_RSA_PKCS_OAEP_SOURCE_TYPE_PTR является указателем на **CK_RSA_PKCS_OAEP_SOURCE_TYPE**.

♦ **CK_RSA_PKCS_OAEP_PARAMS; CK_RSA_PKCS_OAEP_PARAMS_PTR**

CK_RSA_PKCS_OAEP_PARAMS является структурой, предоставляющей параметры для механизма **CKM_RSA_PKCS_OAEP**. Данная структура определяется следующим образом:

```
typedef struct CK_RSA_PKCS_OAEP_PARAMS {
    CK_MECHANISM_TYPE hashAlg;
    CK_RSA_PKCS_MGF_TYPE mgf;
    CK_RSA_PKCS_OAEP_SOURCE_TYPE source;
    CK_VOID_PTR pSourceData;
    CK_ULONG ulSourceDataLen;
} CK_RSA_PKCS_OAEP_PARAMS;
```

Поля данной структуры имеют следующие значения:

hashAlg механизм идентификационного наименования (ID) алгоритма создания списка сообщения, используемый для вычисления списка параметра зашифровывания;

mgf функция генерации маски (mask generation function) для применения к зашифровываемому блоку;

source источник параметра зашифровывания;

pSourceData данные, используемые для ввода источника параметра зашифровывания;

ulSourceDataLen длина ввода источника параметра зашифровывания.

CK_RSA_PKCS_OAEP_PARAMS_PTR является указателем на **CK_RSA_PKCS_OAEP_PARAMS**.

12.1.8 PKCS #1 RSA OAEP

Механизм PKCS #1 RSA OAEP, обозначенный как **CKM_RSA_PKCS_OAEP**, представляет собой многоцелевой механизм, основанный на криптосистеме RSA с открытым ключом и форматированием блока по принципу OAEP, описанного в PKCS #1. Этот механизм поддерживает односторонние зашифровывание и расшифровку, а также упаковку ключа и распаковку ключа.

Данный механизм имеет параметр структуры **CK_RSA_PKCS_OAEP_PARAMS**.

Указанный механизм может упаковывать и распаковывать любой секретный ключ соответствующей длины. Конкретный носитель может и не иметь возможности упаковки/распаковки каждого поддерживаемого им ключа соответствующей длины. Для упаковки «вводом» в операцию по зашифровыванию является значение атрибута **CKA_VALUE** ключа, который упаковывается; аналогично и для распаковки. Этот механизм не упаковывает тип ключа или любую иную информации о ключе, за исключением длины ключа; приложение должно производить это отдельно. В частности, в ходе распаковки данный механизм добавляет в восстанавливаемый ключ только атрибуты **CKA_CLASS** и **CKA_VALUE** (и **CKA_VALUE_LEN**, если данный ключ его имеет); другие атрибуты должны быть определены в шаблоне.

Ограничения на типы ключей и длину данных приведены в таблице 40. Для зашифровывания и расшифровки ввод и вывод данных могут начинаться в одном и том же месте памяти. В таблице 40 *k* – длина модуля RSA в битах, а *hLen* – длина вывода алгоритма получения списка сообщения. Этот алгоритм определен полем *hashAlg* структуры **CK_RSA_PKCS_OAEP_PARAMS**.

Таблица 40 – PKCS #1 RSA OAEP: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Encrypt [1]	Открытый ключ RSA	$\leq k\text{-}2\text{-}2hLen$	k
C_Decrypt [1]	Секретный ключ RSA	k	$\leq k\text{-}2\text{-}2hLen$
C_WrapKey	Открытый ключ RSA	$\leq k\text{-}2\text{-}2hLen$	k
C_UnwrapKey	Секретный ключ RSA	k	$\leq k\text{-}2\text{-}2hLen$
[1] Только односторонние операции.			

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* в структуре **CK_MECHANISM_INFO** указывают поддерживаемый диапазон модуля RSA в битах.

12.1.9 Параметры механизма PKCS #1 RSA PSS

♦ **CK_RSA_PKCS_PSS_PARAMS; CK_RSA_PKCS_PSS_PARAMS_PTR**

CK_RSA_PKCS_PSS_PARAMS представляет собой структуру, которая предоставляет параметры механизму **CKM_RSA_PKCS_PSS**. Данная структура описывается следующим образом:

```
typedef struct CK_RSA_PKCS_PSS_PARAMS {
    CK_MECHANISM_TYPE hashAlg;
    CK_RSA_PKCS_MGF_TYPE mgf;
    CK_ULONG sLen;
} CK_RSA_PKCS_PSS_PARAMS;
```

Поля данной структуры имеют следующие значения:

hashAlg хэш-алгоритм, используемый в зашифровывании по схеме PSS; если механизм постановки подписи не включает хэширование сообщения, тогда это значение должно быть механизмом, применяемым приложением для генерирования хэшированного сообщения; если механизм постановки подписи включает хэширование, тогда это значение должно подходить к хэш-алгоритму, определенному механизмом постановки подписи;

mgf функция генерации маски для применения к зашифровываемому блоку;

sLen длина в битах значения salt (salt value), используемого в кодировании по вероятностной схеме PSS; типичными значениями являются длина хэшированного сообщения и ноль.

CK_RSA_PKCS_PSS_PARAMS_PTR является указателем на **CK_RSA_PKCS_PSS_PARAMS**.

12.1.10 PKCS #1 RSA PSS

Механизм PKCS #1 RSA PSS, обозначенный как **CKM_RSA_PKCS_PSS**, представляет собой механизм, основанный на криптосистеме RSA с открытым ключом и форматированием блока по принципу PSS, описанному в PKCS #1. Этот механизм поддерживает односторонние генерацию и верификацию цифровой подписи без восстановления сообщения. Данный механизм соответствует только той части PKCS #1, которая включает форматирование блока и RSA, с заданным значением хэширования; он не вычисляет значение хэширования на сообщении, которое должно быть подписано.

Указанный механизм имеет параметр структуры **CK_RSA_PKCS_PSS_PARAMS**. Поле *sLen* должно быть меньше или равно $k^*\text{-}2\text{-}hLen$, *hLen* – длина ввода в функцию C_Sign или C_Verify. k^* – длина в битах модуля RSA, за исключением случая, когда длина в битах модуля RSA на единицу больше числа, кратного восьми, в этом случае k^* на единицу меньше, чем длина в битах модуля RSA.

Ограничения на типы ключей и длину данных приведены в таблице 41. В таблице 41 *k* – длина RSA в битах.

Таблица 41 – PKCS #1 RSA PSS: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign [1]	Секретный ключ RSA	*Hlen*	*k*
C_Verify [1]	Открытый ключ RSA	*hLen*, *k* [2]	Не определена
[1] Только односторонние операции. [2] Длина данных, длина подписи.			

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* в структуре **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.1.11 ISO /IEC 9796 RSA

Механизм ISO /IEC 9796 RSA, обозначенный как **CKM_RSA_9796**, представляет собой механизм для односторонних процедур постановки подписи и верификации с восстановлением сообщения и без него, основанный на криптосистеме RSA с открытым ключом и форматированием блоков, определенным в ISO /IEC 9796 и в приложении А к нему.

Этот механизм обрабатывает только байтовые строки, в то время как ISO /IEC 9796 действует с битовыми строками. Соответственно выполняются следующие преобразования:

- Данные конвертируются из байтового в битовый формат следующим образом: самый старший по разрядности бит главного байта байтовой строки интерпретируется как крайний левый бит битовой строки, а самый младший по разрядности бит замыкающего байта байтовой строки интерпретируется как крайний правый бит битовой строки (предполагается, что длина данных в битах кратна восьми).
- Подпись конвертируется из битовой строки в байтовую строку посредством заполнения битовой строки слева нулевыми битами в количестве от нуля до семи, пока результирующая длина в битах не станет кратна восьми, после чего результирующая битовая строка конвертируется, как описано выше; из байтовой строки в битовую строку она конвертируется следующим образом: вначале байтовая строка конвертируется как описано выше, затем слева удаляются биты, пока результирующая длина в битах не станет такой же, как длина модуля RSA.

Данный механизм не имеет параметров.

Ограничения на типы ключей, а также на длину данных ввода и вывода приведены в таблице 42. В таблице 42 *k* – длина модуля RSA в битах.

Таблица 42 – ISO /IEC 9796 RSA: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign [1]	Секретный ключ RSA	$\leq \lfloor k/2 \rfloor$	*k*
C_SignRecover	Секретный ключ RSA	$\leq \lfloor k/2 \rfloor$	*k*
C_Verify [1]	Открытый ключ RSA	$\leq \lfloor k/2 \rfloor$, *k* [2]	Не определена
C_VerifyRecover	Открытый ключ RSA	*k*	$\leq \lfloor k/2 \rfloor$

[1] Только односторонние операции.
[2] Длина данных, длина подписи.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* в структуре **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.1.12 X.509 «грубый» RSA

Механизм X.509 «грубый» RSA, обозначенный как **CKM_RSA_X_509**, представляет собой многоцелевой механизм, основанный на криптосистеме RSA с открытым ключом. Он поддерживает одностороннее шифрование и дешифрование; односторонние процедуры постановки подписи и верификации с восстановлением сообщения и без него; упаковку ключа; распаковку ключа. Все эти операции основываются на так называемом грубом RSA, как принято в X.509.

«Грубый» RSA, как определено в настоящем стандарте, зашифровывает байтовую строку посредством преобразования ее вначале в целый наиболее старший по разряду байт с применением возведения в степень механизма «грубого» RSA с последующим преобразованием результата в байтовую строку, начиная с наиболее старшего по разряду байта. Строка ввода, рассматриваемая как целое число, должна быть меньше, чем модуль; строка вывода также меньше, чем модуль.

Данный механизм не имеет параметра.

Указанный механизм может упаковывать и распаковывать любой секретный ключ соответствующей длины. Конкретный носитель может и не поддерживать упаковку/распаковку каждого поддерживаемого им секретного ключа соответствующей длины. Для упаковки «вводом» в операцию по шифрованию является значение атрибута **CKA_VALUE** ключа, который упаковывается; аналогично и для распаковки. Этот механизм не упаковывает тип ключа или любую иную информацию о ключе, за исключением длины ключа; приложение должно производить это отдельно и представлять эти данные при распаковке ключа.

К сожалению, X.509 не определяет, как должно осуществляться заполнение полей для шифрования RSA. Для данного механизма заполнение должно производиться присоединением перед данными открытого текста нулевых байтов. В результате, чтобы зашифровать последовательность открытых байтов b_1, b_2 … b_n ($n \leq k$), Cryptoki создает $P = 2^{n-1}b_1 + 2^{n-2}b_2 + \ldots + b_n$. Это число должно быть меньше, чем модуль RSA. *k*-байтный шифртекст (*k* – длина в байтах модуля RSA) получают посредством возве-

дения P в степень с открытым экспонентом RSA, равным значению по модулю длины модуля RSA. Дешифрование *k*-байтного шифртекста C производится посредством возведения C в степень с секретным экспонентом RSA, равным значению по модулю длины модуля RSA, с последующим представлением результирующего значения как последовательности длиной точно в *k* байт. Если результирующий открытый текст должен быть использован для получения распакованного ключа, тогда независимо от того эти байты берутся *с конца* данной байтовой последовательности.

С технической точки зрения описанные выше процедуры лишь слегка отличаются от конкретных деталей процедур, описанных в X.509.

Осуществление криптографических операций с использованием данного механизма может привести к сообщениям об ошибках CKR_DATA_INVALID (если представлен открытый текст, имеющий ту же длину, что и модуль RSA, и в численном выражении больше или равен модулю RSA) и CKR_ENCRYPTED_DATA_INVALID (если представлен шифртекст, имеющий ту же длину, что и модуль RSA, и в численном выражении больше или равен модулю RSA).

Ограничения на типы ключей, а также на длину данных ввода и вывода приведены в таблице 43. В таблице 43 *k* – длина модуля RSA в битах.

Таблица 43 – X.509 «грубый» RSA: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Encrypt [1]	Открытый ключ RSA	$\leq k$	k
C_Decrypt [1]	Секретный ключ RSA	k	k
C_Sign [1]	Секретный ключ RSA	$\leq k$	k
C_SignRecover	Секретный ключ RSA	$\leq k$	k
C_Verify [1]	Открытый ключ RSA	$\leq k$, k [2]	Не определена
C_VerifyRecover	Открытый ключ RSA	k	k
C_WrapKey	Открытый ключ RSA	$\leq k$	k
C_UnwrapKey	Секретный ключ RSA	k	$\leq k$ (определена в шаблоне)

[1] Только односторонние операции.
[2] Длина данных, длина подписи.

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

Данный механизм нацелен на совместимость с приложениями, которые не соблюдают требования по форматированию блоков, содержащиеся в PKCS #1 или ISO /IEC 9796.

12.1.13 ANSI X9.31 RSA

Механизм ANSI X9.31 RSA, обозначенный как **CKM_RSA_X9_31**, представляет собой механизм односторонней постановки подписей и верификации без восстановления сообщения, основанный на криптосистеме RSA с открытым ключом и на форматировании блоков, как принято в ANSI X9.31.

Данный механизм использует заголовок и поля заполнения хэш-инкапсулирования. Приложением должно использоваться поле управляющей информации, завершающей кадр инкапсулированных данных.

Указанный механизм обрабатывает только байтовые строки, в то время как ANSI X9.31 работает с битовыми строками. Соответственно выполняются следующие преобразования:

- Данные конвертируются из байтового в битовый формат следующим образом: самый старший по разрядности бит главного байта байтовой строки интерпретируется как крайний левый бит битовой строки, а самый младший по разрядности бит замыкающего байта байтовой строки интерпретируется как крайний правый бит битовой строки (предполагается, что длина данных в битах кратна восьми).
- Подпись конвертируется из битовой строки в байтовую строку посредством заполнения битовой строки слева нулевыми битами в количестве от нуля до семи, пока результирующая длина в битах не станет кратна восьми, после чего результирующая битовая строка конвертируется, как описано выше; из байтовой строки в битовую строку она конвертируется следующим образом: вначале байтовая строка конвертируется, как описано выше, затем слева удаляются биты, пока результирующая длина в битах не станет такой же, как длина модуля RSA.

Данный механизм не имеет параметра.

Ограничения на типы ключей, а также на длину данных ввода и вывода приведены в таблице 44. В таблице 44 *k* – длина модуля RSA в битах. Для всех операций величина *k* должна быть больше или равна 128 и кратна 32, как определено в ANSI X9.31.

Таблица 44 – ANSI X9.31 RSA: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign [1]	Секретный ключ RSA	$\leq k$-2	*k*
C_Verify [1]	Открытый ключ RSA	$\leq k$-2, *k* [2]	Не определена
[1] Только односторонние операции. [2] Длина данных, длина подписи.			

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.1.14 Подпись PKCS #1 v1.5 RSA с MD2, MD5, SHA-1, SHA-256, SHA-384, SHA-512, RIPE-MD 128 или RIPE-MD 160

Подпись PKCS #1 v1.5 RSA с механизмом MD2, обозначенным как **CKM_MD2_RSA_PKCS**, осуществляет в одностороннем и многостороннем режиме постановку цифровых подписей и верификационных операций без восстановления сообщения. Эти операции выполняются в соответствии с первоначальным описанием в PKCS #1 v1.5 с идентификатором объекта md2WithRSAEncryption и в соответствии со схемой RSASSA-PKCS1-v1_5 из действующей версии PKCS #1, где лежащей в основе процесса хэш-функцией является MD2.

Аналогично подпись PKCS #1 v1.5 RSA с механизмом MD5, обозначенным как **CKM_MD5_RSA_PKCS**, осуществляет те же операции, описанные в PKCS #1, с идентификатором объекта md5WithRSAEncryption. Подпись PKCS #1 v1.5 RSA с механизмом SHA-1, обозначенным как **CKM_SHA1_RSA_PKCS**, осуществляет те же операции, за исключением того, что она использует хэш-функцию SHA-1 с идентификатором объекта sha1WithRSAEncryption.

Похожим образом, подпись PKCS #1 v1.5 RSA с механизмами SHA-256, SHA-384 и SHA-512, обозначенными как **CKM_SHA256_RSA_PKCS**, **CKM_SHA384_RSA_PKCS** и **CKM_SHA512_RSA_PKCS** соответственно, осуществляет те же операции, используя хэш-функции SHA-256, SHA-384 и SHA-512 с идентификаторами объекта sha256WithRSAEncryption, sha384WithRSAEncryption и sha384WithRSAEncryption соответственно.

Подпись PKCS #1 v1.5 RSA с механизмами RIPEMD-128 или RIPEMD-160, обозначенными как **CKM_RIPEMD128_RSA_PKCS** и **CKM_RIPEMD160_RSA_PKCS** соответственно, выполняет те же операции, используя хэш-функции RIPE-MD 128 и RIPE-MD 160.

Ни один из данных механизмов не имеет параметра.

Ограничения на типы ключей, а также на длину данных ввода и вывода для этих механизмов приведены в таблице 45. В таблице 45 *k* – длина модуля RSA в битах. Для подписей PKCS #1 v1.5 RSA с механизмом MD2 и PKCS #1 v1.5 RSA с механизмом MD5 величина *k* должна быть больше или равна 27; для подписи PKCS #1 v1.5 RSA с механизмом SHA-1 величина *k* должна быть больше или равна 31 и так далее для других лежащих в основе механизмов хэш-функций с условием, что минимальное значение этой величины всегда на 11 бит больше длины значения хэш-функции.

Таблица 45 – PKCS #1 v1.5 RSA Подписи с различными хэш-функциями: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Sign	Секретный ключ RSA	Любая	*k*	Тип блока 01
C_Verify	Открытый ключ RSA	Любая, *k* [2]	Не определена	Тип блока 01
[2] Длина данных, длина подписи.				

Для этих механизмов поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.1.15 Подпись PKCS #1 RSA PSS с SHA-1, SHA-256, SHA-384 или SHA-512

Подпись PKCS #1 RSA PSS с механизмом SHA-1, обозначенным как **CKM_SHA1_RSA_PKCS_PSS**, в одностороннем и многостороннем режиме осуществляет операции постановки цифровых подписей и верификации без восстановления сообщения. Эти операции выполняются в соответствии с описанием в PKCS #1 с идентификатором объекта id-RSASSA-PSS, т. е. аналогично схеме RSASSA-PSS в PKCS #1, где лежащей в основе механизма хэш-функцией является SHA-1.

Подпись PKCS #1 RSA PSS с механизмами SHA-256, SHA-384 и SHA-512, обозначенными как **CKM_SHA256_RSA_PKCS_PSS**, **CKM_SHA384_RSA_PKCS_PSS** и **CKM_SHA512_RSA_PKCS_PSS** соответственно, выполняет те же операции, используя функции хэширования SHA-256, SHA-384 и SHA-512.

Данные механизмы имеют параметр структуры **CK_RSA_PKCS_PSS_PARAMS**. Поле *sLen* должно быть больше или равно *k**-2-*hLen*, где *hLen* – длина в байтах значения, полученного в результате применения хэш-функции. *k** – длина в байтах модуля RSA, за исключением случаев, когда длина модуля RSA в битах на единицу больше числа, кратного восьми. В этих случаях *k** на единицу меньше, чем длина модуля RSA в байтах.

Ограничения на типы ключей, а также на длину данных ввода и вывода для этих механизмов приведены в таблице 46. В таблице 46 *k* – длина модуля RSA в битах.

Таблица 46 – Подписи PKCS #1 RSA PSS с различными хэш-функциями: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign	Секретный ключ RSA	Любая	*k*
C_Verify	Открытый ключ RSA	Любая, *k* [2]	Не определена
[2] Длина данных, длина подписи.			

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.1.16 Подпись ANSI X9.31 RSA с SHA-1

Подпись ANSI X9.31 RSA с механизмом SHA-1, обозначенным как **CKM_SHA1_RSA_X9_31**, в одностороннем и многостороннем режиме осуществляет операции постановки цифровых подписей и верификации без восстановления сообщения. Эти операции выполняются в соответствии с описанием в ANSI X9.31.

Данный механизм не имеет параметра.

Ограничения на типы ключей, а также на длину данных ввода и вывода приведены в таблице 48. В таблице 48 *k* – длина модуля RSA в битах. Для всех операций величина *k* должна быть больше или равна 128 и кратна 32, как определено в ANSI X9.31.

Таблица 47 – Подписи ANSI X9.31 RSA с SHA-1: Длина ключей и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign	Секретный ключ RSA	Любая	*k*
C_Verify	Открытый ключ RSA	любая, *k* [2]	Не определена
[2] Длина данных, длина подписи.			

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров модуля RSA в битах.

12.2 DSA

12.2.1 Определения

Этот раздел определяет тип ключа CKK_DSA как тип CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов DSA.

Механизмы:

```
CKM_DSA_KEY_PAIR_GEN
CKM_DSA
CKM_DSA_SHA1
CKM_DSA_PARAMETER_GEN
CKM_FORTEZZA_TIMESTAMP
```

12.2.2 Объекты открытого ключа DSA

Объекты открытого ключа DSA (класс объекта – **CKO_PUBLIC_KEY,** тип ключа – **CKK_DSA**) хранят открытые ключи DSA. В таблице 48 определены атрибуты объектов открытого ключа DSA в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 48 – Атрибуты объекта открытого ключа DSA

Атрибут	Тип данных	Значение

Атрибут	Тип данных	Значение
CKA_PRIME [1, 3]	Большое целое	Первичное *p* (от 512 до 1024 бит, шагами по 64 бита)
CKA_SUBPRIME [1, 3]	Большое целое	Предпервичное *q* (160 бит)
CKA_BASE [1, 3]	Большое целое	Базовое *g*
CKA_VALUE [1, 4]	Большое целое	Открытое значение *y*
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME**, **CKA_SUBPRIME** и **CKA_BASE** совместно являются «параметрами домена DSA». Для дополнительной информации по ключам DSA см. FIPS PUB 186-2.

Ниже представлен пример шаблона для создания объекта открытого ключа DSA:

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_DSA;
CK_UTF8CHAR label[] = "Объект открытого ключа DSA";
CK_BYTE prime[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_SUBPRIME, subprime, sizeof(subprime)},
  {CKA_BASE, base, sizeof(base)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.2.3 Объекты секретного ключа DSA

Объекты секретного ключа DSA (класс объекта – **CKO_PRIVATE_KEY,** тип ключа – **CKK_DSA**) содержат секретные ключи DSA. В таблице 49 определены атрибуты объекта секретного ключа DSA в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 49 – Атрибуты объекта секретного ключа DSA

Атрибут	Тип данных	Значение
CKA_PRIME [1, 4, 6]	Большое целое	Первичное *p* (от 512 до 1024 бит, шагами по 64 бита)
CKA_SUBPRIME [1, 4, 6]	Большое целое	Предпервичное *q* (160 бит)
CKA_BASE [1, 4, 6]	Большое целое	Базовое *g*
CKA_VALUE [1, 4, 6, 7]	Большое целое	Секретное значение *x*
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME**, **CKA_SUBPRIME** и **CKA_BASE** совместно являются «параметрами домена DSA» (для дополнительной информации по ключам DSA см. FIPS PUB 186-2).

Следует отметить, что при генерации секретного ключа DSA параметры домена DSA в шаблоне этого ключа *не* определены. Это происходит, потому что секретные ключи DSA генерируются лишь как части *пар* ключей DSA, а параметры домена DSA для пары ключей задаются в шаблоне для открытого ключа DSA.

Ниже представлен пример шаблона для создания объекта секретного ключа DSA:

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_DSA;
CK_UTF8CHAR label[] = "Объект секретного ключа DSA";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE prime[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
```

```
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_SUBJECT, subject, sizeof(subject)},
  {CKA_ID, id, sizeof(id)},
  {CKA_SENSITIVE, &true, sizeof(true)},
  {CKA_SIGN, &true, sizeof(true)},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_SUBPRIME, subprime, sizeof(subprime)},
  {CKA_BASE, base, sizeof(base)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.2.4 Объекты параметра домена DSA

Объекты параметра домена DSA (класс объекта – **CKO_DOMAIN_PARAMETERS,** тип ключа – **CKK_DSA**) содержат параметры домена DSA. В таблице 50 определены атрибуты объекта параметра домена DSA в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 50 – Атрибуты объекта параметра домена DSA

Атрибут	Тип данных	Значение
CKA_PRIME [1, 4]	Большое целое	Первичное *p* (от 512 до 1024 бит, шагами по 64 бита)
CKA_SUBPRIME [1, 4]	Большое целое	Предпервичное *q* (160 бит)
CKA_BASE [1, 4]	Большое целое	Базовое *g*
CKA_PRIME_BITS [2, 3]	CK_ULONG	Длина первичного значения
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME**, **CKA_SUBPRIME** и **CKA_BASE** совместно являются «параметрами домена DSA» (для дополнительной информации по ключам DSA см. FIPS PUB 186-2).

Ниже представлен пример шаблона для создания объекта параметра домена DSA:

```
CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
CK_KEY_TYPE keyType = CKK_DSA;
CK_UTF8CHAR label[] = "Объект параметра домена DSA";
CK_BYTE prime[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE base[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_SUBPRIME, subprime, sizeof(subprime)},
  {CKA_BASE, base, sizeof(base)},
};
```

12.2.5 Генерация пары ключей DSA

Механизм генерации пары ключей DSA, обозначенный как **CKM_DSA_KEY_PAIR_GEN**, представляет собой механизм генерации пары ключей, основанный на алгоритме цифровой подписи, описанном в FIPS PUB 186-2.

Данный механизм не имеет параметра.

Этот механизм генерирует пары, состоящие из открытого и секретного ключей DSA отдельно с первичным, предпервичным и базовым значениями, как определено в атрибутах **CKA_PRIME**, **CKA_SUBPRIME** и **CKA_BASE** шаблона открытого ключа.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому открытому ключу, а также атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_SUBPRIME**, **CKA_BASE** и **CKA_VALUE** к новому секретному ключу. Другие атрибуты, поддерживаемые открытым и секретным типами ключей DSA (в частности, флаги, указывающие, какие функции поддерживает данный ключ), могут также быть заданы в шаблонах ключей, либо им могут быть приданы начальные значения по умолчанию.

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения DSA в битах.

12.2.6 Генерация параметра домена DSA

Механизм генерации параметра домена DSA, обозначенный как **CKM_DSA_PARAMETER_GEN**, представляет собой механизм генерации параметра домена, основанный на алгоритме цифровой подписи, описанном в FIPS PUB 186-2.

Данный механизм не имеет параметра.

Этот механизм генерирует параметры домена DSA с определенной первичной длиной в битах, как определено в атрибуте шаблона **CKA_PRIME_BITS**.

Указанный механизм добавляет к новому объекту атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_SUBPRIME**, **CKA_BASE** и **CKA_PRIME_BITS**. Другие атрибуты, поддерживаемые различными типами параметров домена DSA, могут также быть заданы в шаблоне, либо им могут быть приданы начальные значения по умолчанию.

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения DSA в битах.

12.2.7 DSA без хэширования

Механизм DSA без хэширования, обозначенный как **CKM_DSA**, представляет собой механизм односторонней постановки электронных подписей и верификации, основанный на алгоритме цифровой подписи, описанном в FIPS PUB 186-2 (данный механизм соответствует лишь той части DSA, которая обрабатывает 20-байтовое значение результата хэширования; механизм не вычисляет значение этого результата).

Используемая для целей этого механизма электронная подпись DSA представляет собой 40-байтную строку, соответствующую последовательности значений DSA *r* и *s*, каждое из которых первым представило наиболее старший по разряду байт.

Данный механизм не имеет параметра.

Ограничения на типы ключа и длину данных приведены в таблице 51.

Таблица 51 – DSA: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign [1]	Секретный ключ DSA	20	40
C_Verify [1]	Открытый ключ DSA	20, 40 [2]	Не определена
[1] Только односторонние операции. [2] Длина данных, длина подписи.			

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения DSA в битах.

12.2.8 DSA посредством SHA-1

Механизм DSA посредством SHA-1, обозначенный как **CKM_DSA_SHA1**, представляет собой механизм для односторонней и многосторонней постановки электронной подписи и верификации, основанный на алгоритме цифровой подписи, описанном в FIPS PUB 186-2. Этот механизм вычисляет все характеристики DSA, включая хэширование посредством SHA-1.

Используемая для целей этого механизма электронная подпись DSA представляет собой 40-байтную строку, соответствующую последовательности значений DSA *r* и *s*, каждое из которых первым представило наиболее старший по разряду байт.

Данный механизм не имеет параметра.

Ограничения на типы ключа и длину данных приведены в таблице 52.

Таблица 52 – DSA посредством SHA-1: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign	Секретный ключ DSA	любая	40
C_Verify	Открытый ключ DSA	любая, 40 [2]	Не определена
[2] Длина данных, длина подписи.			

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения DSA в битах.

12.2.9 Временная метка FORTEZZA

Механизм временнòй метки FORTEZZA, обозначенный как **CKM_FORTEZZA_TIMESTAMP**, представляет собой механизм для односторонней постановки подписи и верификации. Подписи, которые производит и верифицирует этот механизм, представляют собой цифровые подписи DSA, наложенные на предоставленное значение результата хэширования и на текущее время.

Данный механизм не имеет параметров.

Ограничения на типы ключей и длину данных приведены в таблице 53. Ввод и вывод данных может начинаться с одной и той же позиции в памяти.

Таблица 53 – Временная метка FORTEZZA: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign [1]	Секретный ключ DSA	20	40
C_Verify [1]	Открытый ключ DSA	20, 40 [2]	Не определена
[1] Только односторонние операции. [2] Длина данных, длина подписи.			

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения DSA в битах.

12.3 Эллиптическая кривая

Представленная в настоящем стандарте криптосистема «Эллиптическая кривая» (Elliptic Curve, EC), также имеющая отношение к ECDSA, представляет собой криптосистему, описанную в стандартах ANSI X9.62 и X9.63, разработанных рабочей группой ANSI X9F1.

Таблица 54 – Информационные флаги механизма

CKF_EC_F_P	0x00100000	Действительно, если механизм может быть использован с параметрами домена EC с F_p
CKF_EC_F_2M	0x00200000	Действительно, если механизм может быть использован с параметрами домена EC с F_{2^m}
CKF_EC_ECPARAMETERS	0x00400000	Действительно, если механизм может быть использован с параметрами варианта **ecParameters**
CKF_EC_NAMEDCURVE	0x00800000	Действительно, если механизм может быть использован с параметрами варианта **namedCurve**
CKF_EC_UNCOMPRESS	0x01000000	Верно, если механизм может быть использован с несжатой вершиной эллиптической кривой
CKF_EC_COMPRESS	0x02000000	Верно, если механизм может быть использован со сжатой вершиной эллиптической кривой

В указанных стандартах определены две отличающиеся разновидности EC:

1. EC, использующая поле с нечетным простым числом элементов (т. е. замыкающее поле F_p);
2. EC, использующая поле с характеристикой два (т. е. замыкающее F_{2^m}).

Ключ EC в системе Cryptoki содержит информацию о том, для какой разновидности EC он предназначен. Предпочтительно, чтобы библиотека Cryptoki, которая может задействовать механизмы EC, была в состоянии выполнять операции с двумя разновидностями EC, хотя категоричного требования об этом нет. Флаг **CKF_EC_F_P** структуры **CK_MECHANISM_INFO** указывает библиотеку Cryptoki, поддерживающую ключи EC keys с F_p, в то время как флаг **CKF_EC_F_2M** указывает библиотеку

Cryptoki, поддерживающую ключи EC с F_{2^m}. Библиотека Cryptoki, которая может задействовать механизмы EC, должна установить один из двух либо оба этих флага для каждого из механизмов EC.

В этих описаниях есть также три метода представления для задания параметров домена любого ключа EC. В Cryptoki поддерживаются только варианты **cParameters** и **namedCurve**. Флаг **CKF_EC_ECPARAMETERS** структуры **CK_MECHANISM_INFO** указывает библиотеку Cryptoki, поддерживающую вариант **ecParameters,** в то время как флаг **CKF_EC_NAMEDCURVE** указывает на библиотеку Cryptoki, поддерживающую вариант **namedCurve**. Библиотека Cryptoki, которая может задействовать механизмы EC, должна установить один из двух либо оба эти флага для каждого из механизмов EC.

В этих описаниях открытый ключ EC (т. е. точка *Q* эллиптической кривой) или базовая точка *G* при использовании варианта **ecParameters** могут быть представлены как строка из восьмизначных групп в сжатой или в несжатой форме. Флаг **CKF_EC_UNCOMPRESS** структуры **CK_MECHANISM_INFO**
указывает библиотеку Cryptoki, поддерживающую несжатую форму, в то время как флаг **CKF_EC_COMPRESS** указывает на библиотеку Cryptoki, поддерживающую сжатую форму. Библиотека Cryptoki, которая может задействовать механизмы EC, должна установить один из двух либо оба этих флага для каждого из механизмов EC.

Следует отметить, что в случае применения реализации библиотеки Cryptoki, поддерживающей EC лишь одной разновидности, одной формы представления параметров домена или одной формы, можно столкнуться с трудностями при попытке достичь совместимости с другими реализациями.

Если будет сделана попытка создать, сгенерировать, преобразовать либо распаковать ключ EC неподдерживаемой разновидности (или поддерживаемой разновидности, но неподдерживаемого размера), такая попытка потерпит неудачу с выдачей кода ошибки CKR_TEMPLATE_INCONSISTENT. Если будет сделана попытка создать, сгенерировать, преобразовать либо распаковать ключ EC с неверной или неподдерживаемой формой представления параметров домена, такая попытка потерпит неудачу с выдачей кода ошибки CKR_DOMAIN_PARAMS_INVALID. Если будет сделана попытка создать, сгенерировать, преобразовать либо распаковать ключ EC с неподдерживаемой формой представления, такая попытка потерпит неудачу с выдачей кода ошибки CKR_TEMPLATE_INCONSISTENT.

12.3.1 Подписи EC

Применяемая для реализации целей данного механизма подпись ECDSA представляет собой строку из восьмизначных групп заданной длины – большей или равной двукратной величине восьмизначных групп *nLen*, где *nLen* является выраженной в восьмизначных группах длиной порядка базовой точки *n*. Применяемые для постановки подписи восьмизначные группы соответствуют последовательности значений ECDSA *r* и *s*, оба эти значения представлены как строки из восьмизначных групп одинаковой длины не более чем *nLen*, начиная с наиболее старшего по разряду байта. Если *r* и *s* имеют различную длину в восьмизначных группах, та из них, которая оказалась короче, должна быть дополнена сначала нулевыми восьмизначными группами, т. е. таким образом, чтобы длины в восьмизначных группах были одинаковыми. Первая половина подписи представляет собой *r*, а вторая половина – *s*. Для подписей, созданных носителем, длина результирующей подписи всегда составляет 2*nLen*. Для подписей, переданных носителю для верификации, подпись может быть меньшей длины, но она должна быть
составлена по правилам, описанным выше.

Если длина результата хэширования больше, чем длина в битах значения *n*, тогда будут использоваться только биты из левой части результата хэширования, на длину, не превышающую размер *n*.

Примечание – Для различных приложений рекомендуется, если возможно, кодировать подпись как строку из восьмизначных групп длиной, вдвое превышающей *nLen*. Это обеспечит работу приложения с модулями PKCS#11, которые применялись на основе более ранних версий данного документа. Эти более ранние версии требовали, чтобы все подписи имели длину, вдвое превышающую *nLen*. Может оказаться невозможным кодировать подпись с максимальной длиной, вдвое превышающей *nLen*, если данное приложение просто получает целые значения *r* и *s* (т. е. без нулей впереди), но не имеет информации о порядке базовой точки *n*, поскольку *r* и *s* могут иметь любое значение между нулем и порядком базовой точки *n*.

12.3.2 Определения

Этот раздел определяет типы ключа CKK_ECDSA, CKK_EC и CKK_DSA как тип CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
Примечание: CKM_ECDSA_KEY_PAIR_GEN исключена из числа рекомендованных в v2.11
CKM_ECDSA_KEY_PAIR_GEN
CKM_EC_KEY_PAIR_GEN
```

```
CKM_ECDSA
CKM_ECDSA_SHA1
CKM_ECDH1_DERIVE
CKM_ECDH1_COFACTOR_DERIVE
CKM_ECMQV_DERIVE
```

12.3.3 Объекты открытого ключа ECDSA

Объекты открытого ключа криптосистемы «Эллиптическая кривая» (Elliptic Curve, EC), относящиеся также к ECDSA (класс объекта – **CKO_PUBLIC_KEY,** тип ключа – **CKK_EC** или **CKK_ECDSA**), содержат открытые ключи EC. В таблице 55 приведены атрибуты открытого ключа EC в дополнение к общим атрибутам, установленным для данного класса объектов.

Таблица 55 – Атрибуты объектов открытого ключа криптосистемы «Эллиптическая кривая»

Атрибут	Тип данных	Значение
CKA_EC_PARAMS [1, 3] (CKA_ECDSA_PARAMS)	Совокупность байтов	DER-кодирование значения ANSI X9.62 `Parameters`
CKA_EC_POINT [1, 4]	Совокупность байтов	DER-кодирование значения *Q* ANSI X9.62 `ECPoint`
См. сноски в таблице 15.		

Значение атрибута **CKA_EC_PARAMS** или **CKA_ECDSA_PARAMS** известно как «параметры домена EC» и определено в ANSI X9.62 как выбор из трех способов представления параметра, записанных следующим образом:

```
Parameters ::= CHOICE {
  ecParameters ECParameters,
  namedCurve         CURVES.&id({CurveNames}),
  implicitlyCA NULL
}
```

Это позволяет сделать подробное описание всех требуемых значений, применяя вариант **ecParameters**, использовать **namedCurve** как заместитель идентификатора объекта для определенного набора параметров домена эллиптической кривой или **implicitlyCA** для указания на то, что эти параметры домена открыто задаются где-то в другом месте. Рекомендуется использование **namedCurve** при выборе варианта **ecParameters**. Вариант **implicitlyCA** не должен использоваться в Cryptoki.

Ниже приводится пример шаблона для создания объекта открытого ключа EC (ECDSA):

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_EC;
CK_UTF8CHAR label[] = "Объекты открытого ключа криптосистемы EC";
CK_BYTE ecParams[] = {...};
CK_BYTE ecPoint[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_EC_PARAMS, ecParams, sizeof(ecParams)},
  {CKA_EC_POINT, ecPoint, sizeof(ecPoint)}
};
```

12.3.4 Объекты секретного ключа криптосистемы «Эллиптическая кривая»

Объекты секретного ключа криптосистемы «Эллиптическая кривая» (Elliptic Curve, EC), относящиеся также и к ECDSA (класс объекта – **CKO_PRIVATE_KEY,** тип ключа – **CKK_EC** или **CKK_ECDSA**), содержат секретные ключи EC (для дополнительной информации о EC см. 12.3). В таблице 56 преведены атрибуты секретного ключа EC в дополнение к общим атрибутам, установленным для данного класса объектов.

Таблица 56 – Атрибуты объектов секретного ключа криптосистемы «Эллиптическая кривая»

Атрибут	Тип данных	Значение
CKA_EC_PARAMS [1, 4, 6] (CKA_ECDSA_PARAMS)	Совокупность байтов	DER-кодирование значения ANSI X9.62 Parameters
CKA_VALUE [1, 4, 6, 7]	Большое целое	Секретное значение *d* ANSI X9.62
См. сноски в таблице 15.		

Значение атрибута **CKA_EC_PARAMS** или **CKA_ECDSA_PARAMS** известно как «параметры домена EC» и определено в ANSI X9.62 как выбор из трех способов представления параметра, записанных следующим образом:

```
Parameters ::= CHOICE {
  ecParameters ECParameters,
  namedCurve         CURVES.&id({CurveNames}),
  implicitlyCA NULL
}
```

Это позволяет сделать подробное описание всех требуемых значений, применяя вариант **ecParameters**, использовать **namedCurve** как заместитель идентификатора объекта для определенного набора параметров домена эллиптической кривой или **implicitlyCA** для указания на то, что эти параметры домена открыто задаются где-то в другом месте. Рекомендуется использование **namedCurve** при выборе варианта **ecParameters**. Вариант **implicitlyCA** не должен использоваться в Cryptoki.

Следует отметить, что при генерации секретного ключа EC параметры домена EC в шаблоне этого ключа *не* определены. Это происходит, потому что секретные ключи DSA генерируются лишь как части *пар* ключей EC, а параметры домена EC для пары ключей задаются в шаблоне для открытого ключа EC.

Ниже приводится пример шаблона для создания объекта секретного ключа EC (ECDSA):

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_EC;
CK_UTF8CHAR label[] = "Объект секретного ключа EC";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE ecParams[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_SUBJECT, subject, sizeof(subject)},
  {CKA_ID, id, sizeof(id)},
  {CKA_SENSITIVE, &true, sizeof(true)},
  {CKA_DERIVE, &true, sizeof(true)},
  {CKA_EC_PARAMS, ecParams, sizeof(ecParams)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.3.5 Генерация пар ключей криптосистемы «Эллиптическая кривая»

Механизм генерации пар ключей криптосистемы «Эллиптическая кривая» (Elliptic Curve, EC), относящийся также и к ECDSA и обозначенный как **CKM_EC_KEY_PAIR_GEN** или **CKM_ECDSA_KEY_PAIR_GEN**, представляет собой механизм генерации пар ключей для EC.

Данный механизм не имеет параметра.

Этот механизм генерирует пары из открытого и секретного ключей в соответствии с параметрами определенного домена EC, как этот определено в атрибуте **CKA_EC_PARAMS** или **CKA_ECDSA_PARAMS** для шаблона открытого ключа. Следует отметить, что данная версия Cryptoki не содержит механизм для генерации параметров этого домена EC.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_EC_POINT** в новый открытый ключ, а атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_EC_PARAMS** или **CKA_ECDSA_PARAMS,** а также **CKA_CKA_VALUE** в новый секретный ключ. Другие атрибуты, поддерживаемые различными типами открытых и секретных ключей EC (в частности, флаги, указывающие, какие функции поддерживают эти ключи), могут также быть заданы в шаблоне, либо им могут быть приданы начальные значения по умолчанию.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют, соответственно, минимальное и максимальное количество бит в размерах поля. Например, если библиотека Cryptoki поддерживает только ECDSA с использованием поля с характеристикой 2, которое включает от 2^{200} до 2^{300} элементов, тогда *ulMinKeySize* = 201, а *ulMaxKeySize* = 301 (при записи в двоичной системе число 2^{200} состоит из одного единичного бита и 200 последующих нулевых бит. Таким образом, это число длиной в 201 бит. Аналогично 2^{300} – число длиной в 301 бит).

12.3.6 ECDSA без хэширования

Описание кодирования подписи содержится в разделе 12.3.1.

Механизм ECDSA без хэширования, обозначенный как **CKM_ECDSA**, представляет собой механизм для односторонних операций постановки подписи и верификации для ECDSA. (Этот механизм соответствует лишь той части ECDSA, которая имеет дело с обработкой значений, полученных в результате хэширования. Эти значения не должны быть длиннее 1024 бит. Данный механизм не вычисляет результат хэширования.)

Данный механизм не имеет параметра.

Ограничения на типы ключа и длину данных приведены в таблице 57.

Таблица 57 – ECDSA: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign [1]	Секретный ключ ECDSA	Любая [3]	2*nLen*
C_Verify [1]	Открытый ключ ECDSA	Любая [3], ≤ 2*nLen* [2]	Не определена
[1] Только односторонние операции. [2] Длина данных, длина подписи. [3] Усекается до необходимого числа бит.			

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют, соответственно, минимальное и максимальное количество бит в размерах поля. Например, если библиотека Cryptoki поддерживает только ECDSA с использованием поля с характеристикой 2, которое включает от 2^{200} до 2^{300} элементов, тогда *ulMinKeySize* = 201, а *ulMaxKeySize* = 301 (при записи в двоичной системе число 2^{200} состоит из одного единичного бита и 200 последующих нулевых бит. Таким образом, это число длиной в 201 бит. Аналогично 2^{300} – число длиной в 301 бит).

12.3.7 ECDSA посредством SHA-1

Описание кодирования подписи содержится в разделе 12.3.1.

Механизм ECDSA посредством SHA-1, обозначенный как **CKM_ECDSA_SHA1**, представляет собой механизм для односторонних и двусторонних операций постановки подписи и верификации для ECDSA. Этот механизм вычисляет все характеристики ECDSA, включая хэширование посредством SHA-1.

Данный механизм не имеет параметра.

Ограничения на типы ключа и длину данных приведены в таблице 58.

Таблица 58 – ECDSA посредством SHA-1: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Sign	Секретный ключ ECDSA	Любая	2*nLen*
C_Verify	Открытый ключ ECDSA	Любая, ≤ 2*nLen* [2]	Не определена
[2] Длина данных, длина подписи.			

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют, соответственно, минимальное и максимальное количество бит в размерах поля. Например, если библиотека Cryptoki поддерживает только ECDSA с использованием поля с характеристикой 2, которое включает от 2^{200} до 2^{300} элементов, тогда *ulMinKeySize* = 201, а *ulMaxKeySize* = 301 (при записи

в двоичной системе число 2^{200} состоит из одного единичного бита и 200 последующих нулевых бит.

Таким образом, это число длиной в 201 бит. Аналогично 2^{300} – число длиной в 301 бит).

12.3.8 Параметры механизма криптосистемы «Эллиптическая кривая»

♦ **CK_EC_KDF_TYPE, CK_EC_KDF_TYPE_PTR**

CK_EC_KDF_TYPE используется для обозначения функции деривации ключа (Key Derivation Function, KDF), применяемой для получения (деривации) данных ключа из совместных секретных данных. Функция деривации ключа будет использоваться по схемам согласования ключей. Она задается следующим образом:

```
typedef CK_ULONG CK_EC_KDF_TYPE;
```

В таблице 59 перечисляются заданные функции.

Таблица 59 – EC: Функции деривации ключа

Идентификатор источника	Значение
CKD_NULL	0x00000001
CKD_ SHA1_KDF	0x00000002

Функция деривации ключа **CKD_NULL** производит грубое вычисление общего секретного значения без применения какой бы то ни было функции деривации ключа, в то время как функция деривации ключа **CKD_SHA1_KDF**, которая основана на SHA-1, получает данные ключа из общего секретного значения, как определено в ANSI X9.63.

CK_EC_KDF_TYPE_PTR является указателем на **CK_EC_KDF_TYPE**.

♦ **CK_ECDH1_DERIVE_PARAMS, CK_ECDH1_DERIVE_PARAMS_PTR**

CK_ECDH1_DERIVE_PARAMS является структурой, которая предоставляет параметры для механизма деривации ключей **CKM_ECDH1_DERIVE** и **CKM_ECDH1_COFACTOR_DERIVE,** причем каждая сторона выдает одну пару ключей. Данная структура задается следующим образом:

```
typedef struct CK_ECDH1_DERIVE_PARAMS {
   CK_EC_KDF_TYPE kdf;
   CK_ULONG ulSharedDataLen;
   CK_BYTE_PTR pSharedData;
   CK_ULONG ulPublicDataLen;
   CK_BYTE_PTR pPublicData;
} CK_ECDH1_DERIVE_PARAMS;
```

Поля данной структуры имеют следующие значения:

kdf функция деривации ключа, примененная к совместному секретному значению;
ulSharedDataLen длина общей информации в байтах;
pSharedData некоторые данные, общие для обеих сторон;
ulPublicDataLen длина в битах открытого ключа EC другой стороны;
pPublicData указатель на значение открытого ключа EC другой стороны.

Если применяется функция деривации ключа **CKD_NULL**, значение *pSharedData* должно быть NULL, а значение *ulSharedDataLen* должно быть ноль. Если применяется функция деривации ключа **CKD_SHA1_KDF**, может быть предложено произвольное значение *pSharedData*, состоящее из некоторых данных, которые обе стороны сделали общими, желая обеспечить одинаковый доступ к общим секретным данным. В противном случае, значение *pSharedData* должно быть NULL, а значение *ulSharedDataLen* должно быть ноль.

CK_ECDH1_DERIVE_PARAMS_PTR является указателем на **CK_ECDH1_DERIVE_PARAMS**.

♦ **CK_ ECMQV _DERIVE_PARAMS, CK_ ECMQV _DERIVE_PARAMS_PTR**

CK_ ECMQV _DERIVE_PARAMS является структурой, которая предоставляет параметры механизму деривации ключа **CKM_ECMQV_DERIVE.** Каждая из сторон направляет в эту структуру две пары ключей. Данная структура задается следующим образом:

```
typedef struct CK_ECMQV_DERIVE_PARAMS {
   CK_EC_KDF_TYPE kdf;
   CK_ULONG ulSharedDataLen;
   CK_BYTE_PTR pSharedData;
   CK_ULONG ulPublicDataLen;
```

```
    CK_BYTE_PTR pPublicData;
    CK_ULONG ulPrivateDataLen;
    CK_OBJECT_HANDLE hPrivateData;
    CK_ULONG ulPublicDataLen2;
    CK_BYTE_PTR pPublicData2;
    CK_OBJECT_HANDLE publicKey;
} CK_ECMQV_DERIVE_PARAMS;
```

Поля данной структуры имеют следующие значения:

kdf	функция деривации ключа, применяемая к совместному секретному значению;
ulSharedDataLen	длина совместной информации в байтах;
pSharedData	некоторые данные, общие для обеих сторон;
ulPublicDataLen	длина в байтах первого открытого ключа EC другой стороны;
pPublicData	указатель на значение первого открытого ключа EC другой стороны;
ulPrivateDataLen	длина в байтах второго секретного ключа EC;
hPrivateData	управление ключом для значения второго секретного ключа EC;
ulPublicDataLen2	длина в байтах второго открытого ключа EC второй стороны;
pPublicData2	указатель на значение второго открытого ключа EC второй стороны;
publicKey	управление временным открытым ключом второй стороны.

Если применяется функция деривации ключа **CKD_NULL**, значение *pSharedData* должно быть NULL, а значение *ulSharedDataLen* должно быть ноль. Если применяется функция деривации ключа **CKD_SHA1_KDF**, может быть предложено произвольное значение *pSharedData*, состоящее из некоторых данных, которые обе стороны сделали общими, желая обеспечить одинаковый доступ к общим секретным данным. В противном случае значение *pSharedData* должно быть NULL, а значение *ulSharedDataLen* должно быть ноль.

CK_ECMQV_DERIVE_PARAMS_PTR является указателем на **CK_ECMQV_DERIVE_PARAMS**.

12.3.9 Деривация ключа по принципу эллиптической кривой Диффи – Хеллмана

Механизм деривации ключа по принципу эллиптической кривой Диффи – Хеллмана (The elliptic curve Diffie-Hellman, ECDH), обозначенный как **CKM_ECDH1_DERIVE**, представляет собой механизм деривации ключа, основанный на схеме согласования ключа по принципу эллиптической кривой в версии Диффи – Хеллмана, как это определено в ANSI X9.63, где каждая из сторон представляет по одной паре ключей, используя одни и те же параметры домена.

Данный механизм имеет параметр структуры **CK_ECDH1_DERIVE_PARAMS**.

Данный механизм получает секретное значение, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, как его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют, соответственно, минимальное и максимальное количество бит в размерах поля. Например, если библиотека Cryptoki поддерживает только ECDSA с использованием поля с характеристикой 2, которое включает от 2^{200} до 2^{300} элементов, тогда *ulMinKeySize* = 201, а *ulMaxKeySize* = 301 (при записи в двоичной системе число 2^{200} состоит из одного единичного бита и 200 последующих нулевых бит.
Таким образом, это число длиной в 201 бит. Аналогично, 2^{300} – число длиной в 301 бит).

12.3.10 Деривация по принципу эллиптической кривой Диффи – Хеллмана с алгебраическим дополнением

Механизм деривации ключа по принципу эллиптической кривой Диффи – Хеллмана с алгебраическим дополнением (The elliptic curve Diffie-Hellman (ECDH) with cofactor), обозначенный как **CKM_ECDH1_COFACTOR_DERIVE**, представляет собой механизм деривации ключа, основанный на схеме согласования ключа по принципу эллиптической кривой в версии Диффи – Хеллмана с алгебраическим дополнением, как это определено в ANSI X9.63, где каждая из сторон представляет по одной паре ключей, используя одни и те же параметры домена. Умножение с алгебраическим дополнением эффективно для вычислений и помогает избегать проблем с безопасностью, вызываемых, например, «атаками малой группой».

Данный механизм имеет параметр структуры **CK_ECDH1_DERIVE_PARAMS**.

Указанный механизм получает секретное значение, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют, соответственно, минимальное и максимальное количество бит в размерах поля. Например, если библиотека Cryptoki поддерживает только ECDSA с использованием поля с характеристикой 2, которое включает от 2^{200} до 2^{300} элементов, тогда *ulMinKeySize* = 201, а *ulMaxKeySize* = 301 (при записи в двоичной системе число 2^{200} состоит из одного единичного бита и 200 последующих нулевых бит.

Таким образом, это число длиной в 201 бит. Аналогично 2^{300} – число длиной в 301 бит).

12.3.11 Деривация ключа по принципу эллиптической кривой Менезеса – Ку – Вэнстоуна

Механизм деривации ключа по принципу эллиптической кривой Менезеса – Ку – Вэнстоуна (The elliptic curve Menezes-Qu-Vanstone, ECMQV), обозначенный как **CKM_ECMQV_DERIVE**, представляет собой механизм деривации ключа, основанный на схеме согласования ключа по принципу эллиптической кривой в версии Менезеса – Ку – Вэнстоуна, как это определено в ANSI X9.63, где каждая из сторон представляет по две пары ключей, используя одни и те же параметры домена.

Этот механизм имеет параметр структуры **CK_ECMQV_DERIVE_PARAMS**.

Данный механизм получает секретное значение, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа

атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют, соответственно, минимальное и максимальное количество бит в размерах поля. Например, если библиотека Cryptoki поддерживает только ECDSA с использованием поля с характеристикой 2, которое включает от 2^{200} до 2^{300} элементов, тогда *ulMinKeySize* = 201, а *ulMaxKeySize* = 301 (при записи в двоичной системе число 2^{200} состоит из одного единичного бита и 200 последующих нулевых бит.
Таким образом, это число длиной в 201 бит. Аналогично 2^{300} – число длиной в 301 бит).

12.4 Диффи – Хеллман

12.4.1 Определения

Этот раздел определяет тип ключа CKK_DH как тип CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов DH.

Механизмы:
CKM_DH_PKCS_KEY_PAIR_GEN
CKM_DH_PKCS_DERIVE
CKM_X9_42_DH_KEY_PAIR_GEN
CKM_X9_42_DH_DERIVE
CKM_X9_42_DH_HYBRID_DERIVE
CKM_X9_42_MQV_DERIVE
CKM_DH_PKCS_PARAMETER_GEN
CKM_X9_42_DH_PARAMETER_GEN

12.4.2 Объекты открытого ключа Диффи – Хеллмана

Объекты открытого ключа Диффи – Хеллмана (класс объекта – **CKO_PUBLIC_KEY,** тип ключа – **CKK_DH**) содержат открытые ключи Диффи – Хеллмана. В таблице 60 приведены атрибуты открытого ключа Диффи – Хеллмана в дополнение к общим атрибутам, установленным для данного класса объектов.

Таблица 60 – Атрибуты открытого ключа Диффи – Хеллмана

Атрибут	Тип данных	Значение
CKA_PRIME [1, 3]	Большое целое	Первичное *p*
CKA_BASE [1, 3]	Большое целое	Базовое *g*
CKA_VALUE [1, 4]	Большое целое	Открытое значение *y*
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME** и **CKA_BASE** вместе составляют «параметры домена Диффи – Хеллмана». В зависимости от носителя могут вводиться ограничения на длину компонентов ключа (для дополнительной информации по ключам Диффи – Хеллмана см. PKCS #3).

Ниже представлен образец шаблона для создания объекта открытого ключа Диффи – Хеллмана:

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_DH;
CK_UTF8CHAR label[] = "Объект открытого ключа Диффи - Хеллмана";
CK_BYTE prime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_BASE, base, sizeof(base)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.4.3 Объекты открытого ключа Диффи – Хеллмана X9.42

Объекты открытого ключа Диффи – Хеллмана X9.42 (класс объекта – **CKO_PUBLIC_KEY,** тип ключа – **CKK_X9_42_DH**) содержат открытые ключи Диффи – Хеллмана X9.42. В таблице 61 приведены атрибуты открытого ключа Диффи – Хеллмана X9.42 в дополнение к общим атрибутам, установленным для данного класса объектов.

Таблица 61 – Атрибуты объекта открытого ключа Диффи – Хеллмана X9.42

Атрибут	Тип данных	Значение
CKA_PRIME [1, 3]	Большое целое	Первичное p (≥ 1024 бит, шагами по 256 бит)
CKA_BASE [1, 3]	Большое целое	Базовое g
CKA_SUBPRIME [1, 3]	Большое целое	Предпервичное q (≥ 160 бит)
CKA_VALUE [1, 4]	Большое целое	Открытое значение y
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME**, **CKA_BASE** и **CKA_SUBPRIME** вместе составляют «параметры домена Диффи – Хеллмана X9.42» (для дополнительной информации по ключам Диффи – Хеллмана X9.42 см. настоящий стандарт).

Ниже представлен образец шаблона для создания объекта открытого ключа Диффи – Хеллмана X9.42:

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_X9_42_DH;
CK_UTF8CHAR label[] = "Объект открытого ключа Диффи - Хеллмана X9.42";
CK_BYTE prime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_BASE, base, sizeof(base)},
  {CKA_SUBPRIME, subprime, sizeof(subprime)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.4.4 Объекты секретного ключа Диффи – Хеллмана

Объекты открытого ключа Диффи – Хеллмана (класс объекта – **CKO_PRIVATE_KEY,** тип ключа – **CKK_DH**) содержат секретные ключи Диффи – Хеллмана. В таблице 62 приведены атрибуты секретного ключа Диффи – Хеллмана в дополнение к общим атрибутам, установленным для данного класса объектов.

Таблица 62 – Атрибуты объектов секретного ключа Диффи – Хеллмана

Атрибут	Тип данных	Значение
CKA_PRIME [1, 4, 6]	Большое целое	Первичное p
CKA_BASE [1, 4, 6]	Большое целое	Базовое g
CKA_VALUE [1, 4, 6, 7]	Большое целое	Секретное значение x
CKA_VALUE_BITS [2, 6]	CK_ULONG	Длина в битах секретного значения x
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME** и **CKA_BASE** вместе составляют «параметры домена Диффи – Хеллмана». В зависимости от носителя могут вводиться ограничения на длину компонентов ключа (для дополнительной информации по ключам Диффи – Хеллмана см. PKCS #3).

Следует отметить, что при генерации секретного ключа Диффи – Хеллмана параметры Диффи – Хеллмана *не* задаются в шаблоне данного ключа. Причина состоит в том, что секретные ключи

Диффи – Хеллмана генерируются только как часть *пары* ключей Диффи – Хеллмана, а параметры домена Диффи – Хеллмана для этой пары определены в шаблоне для открытого ключа Диффи – Хеллмана.

Ниже представлен образец шаблона для создания объекта открытого ключа Диффи – Хеллмана:

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_DH;
CK_UTF8CHAR label[] = "Объект открытого ключа Диффи - Хеллмана";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE prime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_SUBJECT, subject, sizeof(subject)},
  {CKA_ID, id, sizeof(id)},
  {CKA_SENSITIVE, &true, sizeof(true)},
  {CKA_DERIVE, &true, sizeof(true)},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_BASE, base, sizeof(base)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.4.5 Объекты секретного ключа Диффи – Хеллмана X9.42

Объекты секретного ключа Диффи – Хеллмана X9.42 (класс объекта – **CKO_PRIVATE,** тип ключа – **CKK_X9_42_DH**) содержат секретные ключи Диффи – Хеллмана X9.42. В таблице 63 приведены атрибуты секретного ключа Диффи – Хеллмана X9.42 в дополнение к общим атрибутам, установленным для данного класса объектов.

Таблица 63 – Атрибуты объекта секретного ключа Диффи – Хеллмана X9.42

Атрибут	Тип данных	Значение
CKA_PRIME [1, 4, 6]	Большое целое	Первичное p (≥ 1024 бит, шагами по 256 бит)
CKA_BASE [1, 4, 6]	Большое целое	Базовое g
CKA_SUBPRIME [1, 4, 6]	Большое целое	Предпервичное q (≥ 160 бит)
CKA_VALUE [1, 4, 6, 7]	Большое целое	Секретное значение x
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME**, **CKA_BASE** и **CKA_SUBPRIME** вместе составляют «параметры домена Диффи – Хеллмана X9.42». В зависимости от носителя могут вводиться ограничения на длину компонентов ключа (для дополнительной информации по ключам Диффи – Хеллмана X9.42 см. ANSI X9.42).

Следует отметить, что при генерации секретного ключа Диффи – Хеллмана X9.42 параметры Диффи – Хеллмана *не* задаются в шаблоне данного ключа. Причина состоит в том, что секретные ключи Диффи – Хеллмана X9.42 генерируются только как часть *пары* ключей Диффи – Хеллмана X9.42, а параметры домена Диффи – Хеллмана X9.42 для этой пары определены в шаблоне для открытого ключа Диффи – Хеллмана X9.42.

Ниже представлен образец шаблона для создания объекта секретного ключа Диффи – Хеллмана X9.42:

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_X9_42_DH;
CK_UTF8CHAR label[] = "Объект секретного ключа Диффи - Хеллмана X9.42";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE prime[] = {...};
```

```
CK_BYTE base[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_SUBJECT, subject, sizeof(subject)},
  {CKA_ID, id, sizeof(id)},
  {CKA_SENSITIVE, &true, sizeof(true)},
  {CKA_DERIVE, &true, sizeof(true)},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_BASE, base, sizeof(base)},
  {CKA_SUBPRIME, subprime, sizeof(subprime)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.4.6 Объекты параметров домена Диффи – Хеллмана

Объекты параметра домена Диффи – Хеллмана (класс объекта – **CKO_DOMAIN_PARAMETERS,** тип ключа – **CKK_ DH**) содержат параметры домена Диффи – Хеллмана. В таблице 64 указаны атрибуты параметров домена Диффи – Хеллмана в дополнение к общим атрибутам, установленным для данного класса объектов.

Таблица 64 – Атрибуты объекта параметров домена Диффи – Хеллмана

Атрибут	Тип данных	Значение
CKA_PRIME [1, 4]	Большое целое	Первичное *p*
CKA_BASE [1, 4]	Большое целое	Базовое *g*
CKA_PRIME_BITS [2, 3]	CK_ULONG	Длина первичного значения
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME** и **CKA_BASE** вместе составляют «параметры домена Диффи – Хеллмана». В зависимости от носителя могут вводиться ограничения на длину компонентов ключа (для дополнительной информации по параметрам домена Диффи – Хеллмана см. PKCS #3).

Ниже представлен образец шаблона для создания объекта параметра домена Диффи – Хеллмана:

```
CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
CK_KEY_TYPE keyType = CKK_DH;
CK_UTF8CHAR label[] = "Объект параметра домена Диффи - Хеллмана";
CK_BYTE prime[] = {...};
CK_BYTE base[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_BASE, base, sizeof(base)},
};
```

12.4.7 Объекты параметров домена Диффи – Хеллмана X9.42

Объекты параметров домена Диффи – Хеллмана X9.42 (класс объекта – **CKO_DOMAIN_ PARAMETERS,** тип ключа – **CKK_X9_42_DH**) содержат параметры домена Диффи – Хеллмана X9.42. В таблице 65 приведены атрибуты параметров домена Диффи – Хеллмана X9.42 в дополнение к общим атрибутам, установленным для данного класса объектов.

Таблица 65 – Атрибуты объекта параметров домена Диффи – Хеллмана X9.42

Атрибут	Тип данных	Значение
CKA_PRIME [1, 4]	Большое целое	Первичное *p* (≥ 1024 бит, шагами по 256 бит)
CKA_BASE [1, 4]	Большое целое	Базовое *g*
CKA_SUBPRIME [1, 4]	Большое целое	Предпервичное *q* (≥ 160 бит)
CKA_PRIME_BITS [2, 3]	CK_ULONG	Длина первичного значения
CKA_SUBPRIME_BITS [2, 3]	CK_ULONG	Длина предпервичного значения
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME**, **CKA_BASE** и **CKA_SUBPRIME** вместе составляют «параметры домена Диффи – Хеллмана X9.42». В зависимости от носителя могут вводиться ограничения на длину компонентов ключа (для дополнительной информации по параметрам домена Диффи – Хеллмана X9.42 см. ANSI X9.42).

Ниже представлен образец шаблона для создания объекта параметров домена Диффи – Хеллмана X9.42:

```
CK_OBJECT_CLASS class = CKO_DOMAIN_PARAMETERS;
CK_KEY_TYPE keyType = CKK_X9_42_DH;
CK_UTF8CHAR label[] = "Объект параметров домена Диффи - Хеллмана X9.42";
CK_BYTE prime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE subprime[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_BASE, base, sizeof(base)},
  {CKA_SUBPRIME, subprime, sizeof(subprime)},
};
```

12.4.8 Генерация пары ключей Диффи – Хеллмана PKCS #3

Механизм генерации пары ключей Диффи – Хеллмана PKCS #3, обозначенный как **CKM_DH_PKCS_KEY_PAIR_GEN**, представляет собой механизм генерации пары ключей, основанный на согласовании ключей Диффи – Хеллмана, как это определено в PKCS #3. Это то, что в PKCS #3 названо «Фаза I».

Данный механизм не имеет параметра.

Этот механизм генерирует пары, состоящие из открытого и секретного ключей DSA отдельно с первичным и базовым значениями, как определено в атрибутах **CKA_PRIME** и **CKA_BASE** шаблона открытого ключа. Если задан атрибут **CKA_VALUE_BITS** секретного ключа, механизм ограничивает длину в битах секретной величины, как описано в PKCS #3.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому открытому ключу, а также атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_BASE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_BITS**, если он уже не задан в шаблоне) к новому секретному ключу. Другие атрибуты, которые требуются для типов открытого и секретного ключей Диффи – Хеллмана, должны быть указаны в шаблонах.

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения Диффи – Хеллмана в битах.

12.4.9 Генерация параметра домена Диффи – Хеллмана PKCS #3

Механизм генерации параметра домена Диффи – Хеллмана PKCS #3, обозначенный как **CKM_DH_PKCS_PARAMETER_GEN**, является механизмом генерации параметра домена, основанным на согласовании ключей Диффи – Хеллмана, определенном в PKCS #3.

Данный механизм не имеет параметра.

Этот механизм генерирует параметры домена Диффи – Хеллмана с определенной первичной длиной в битах, как определено в атрибуте шаблона **CKA_PRIME_BITS**.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**,

CKA_BASE и **CKA_PRIME_BITS** к новому объекту. В шаблоне могут также быть заданы другие атрибуты, поддерживаемые различными типами параметра домена Диффи – Хеллмана, либо им могут быть приданы
начальные значения по умолчанию.

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения Диффи – Хеллмана в битах.

12.4.10 Деривация ключей Диффи – Хеллмана PKCS #3

Механизм деривации ключей Диффи – Хеллмана PKCS #3, обозначенный как **CKM_DH_PKCS_DERIVE**, представляет собой механизм деривации ключей, основанный на согласовании ключей Диффи – Хеллмана, как это определено в PKCS #3. Это то, что в PKCS #3 названо «Фаза II».

Данный механизм имеет параметр, который представляет собой открытое значение другой стороны в протоколе согласования ключей, представленное как «Большое целое» системы Cryptoki (т. е. последовательность байтов, где старший по разрядности байт идет первым).

Этот механизм получает секретный ключ из секретного ключа Диффи – Хеллмана и открытого ключа другой стороны. Он вычисляет секретное значение Диффи – Хеллмана из открытого значения и секретного ключа в соответствии с PKCS #3, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа [1]:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют, соответственно, минимальное и максимальное количество бит в размерах поля.

12.4.11 Параметры механизма Диффи – Хеллмана X9.42

♦ **CK_X9_42_DH_KDF_TYPE, CK_X9_42_DH_KDF_TYPE_PTR**

CK_X9_42_DH_KDF_TYPE используется для обозначения функции деривации ключа (Key Derivation Function, KDF), применяемой для получения (деривации) данных ключа из совместных секретных данных. Функция деривации ключа будет использоваться по схемам согласования ключей Диффи – Хеллмана X9.42.

Она задается следующим образом:

```
typedef CK_ULONG CK_X9_42_DH_KDF_TYPE;
```

В таблице 66 перечислены заданные функции.

Таблица 66 – Функции деривации ключа Диффи – Хеллмана X9.42

Идентификатор источника	Значение
CKD_NULL	0x00000001
CKD_SHA1_KDF_ASN1	0x00000003
CKD_SHA1_KDF_CONCATENATE	0x00000004

[1] Правила, касающиеся атрибутов **CKA_SENSITIVE**, **CKA_EXTRACTABLE**, **CKA_ALWAYS_SENSITIVE** и **CKA_NEVER_EXTRACTABLE,** в версии 2.11 были изменены с целью приведения их в соответствие с политикой, применяемой в отношении других механизмов деривации ключей, таких как **CKM_SSL3_MASTER_KEY_DERIVE**.

Функция деривации ключа **CKD_NULL** производит грубое вычисление общего секретного значения без применения какой бы то ни было функции деривации ключа, в то время как функции деривации ключа **CKD_SHA1_KDF_ASN1** и **CKD_SHA1_KDF_CONCATENATE**, которые основаны на SHA-1, получают данные ключа из общего секретного значения, как определено в ANSI X9.42.

CK_X9_42_DH_KDF_TYPE_PTR является указателем на **CK_X9_42_DH_KDF_TYPE**.

♦ **CK_X9_42_DH1_DERIVE_PARAMS, CK_X9_42_DH1_DERIVE_PARAMS_PTR**

CK_X9_42_DH1_DERIVE_PARAMS является структурой, которая предоставляет параметры для механизма деривации ключей **CKM_X9_42_DH_DERIVE,** причем каждая сторона выдает одну пару ключей. Данная структура задается следующим образом:

```
typedef struct CK_X9_42_DH1_DERIVE_PARAMS {
   CK_X9_42_DH_KDF_TYPE kdf;
   CK_ULONG ulOtherInfoLen;
   CK_BYTE_PTR pOtherInfo;
   CK_ULONG ulPublicDataLen;
   CK_BYTE_PTR pPublicData;
} CK_X9_42_DH1_DERIVE_PARAMS;
```

Поля данной структуры имеют следующие значения:

kdf функция деривации ключа, примененная к совместному секретному значению;
ulOtherInfoLen длина другой информации в байтах;
pOtherInfo некоторые данные, общие для обеих сторон;
ulPublicDataLen длина в байтах открытого ключа Диффи – Хеллмана X9.42 другой стороны;
pPublicData указатель на значение открытого ключа Диффи – Хеллмана X9.42 другой стороны.

Если применяется функция деривации ключа **CKD_NULL**, значение *pOtherInfo* должно быть NULL, а значение *ulOtherInfoLen* должно быть ноль. Если применяется функция деривации ключа **CKD_SHA1_ KDF_ASN1**, может быть предложено значение *pOtherInfo*, содержащее восьмизначную строку, определенную в стандарте кодирования ASN.1 DER, и состоящее из некоторых обязательных и необязательных данных, которые обе стороны сделали общими, желая обеспечить одинаковый доступ к общим секретным данным. Если применяется функция деривации ключа **CKD_SHA1_KDF_CONCATENATE**, может быть предложено необязательное значение *pOtherInfo*, состоящее из некоторых данных, которые обе стороны сделали общими, желая обеспечить одинаковый доступ к общим секретным данным. В противном случае значение *pOtherInfo* должно быть NULL, а значение *ulOtherInfoLen* должно быть ноль.

CK_X9_42_DH1_DERIVE_PARAMS_PTR являетя указателем на **CK_X9_42_DH1_DERIVE_PARAMS**.

♦ **CK_X9_42_DH2_DERIVE_PARAMS, CK_X9_42_DH2_DERIVE_PARAMS_PTR**

CK_X9_42_DH2_DERIVE_PARAMS представляет собой структуру, предоставляющую параметры для механизмов деривации ключа **CKM_X9_42_DH_HYBRID_DERIVE** и **CKM_X9_42_MQV_DERIVE,** где каждая сторона выдает две пары ключей. Данная структура задается следующим образом:

```
typedef struct CK_X9_42_DH2_DERIVE_PARAMS {
   CK_X9_42_DH_KDF_TYPE kdf;
   CK_ULONG ulOtherInfoLen;
   CK_BYTE_PTR pOtherInfo;
   CK_ULONG ulPublicDataLen;
   CK_BYTE_PTR pPublicData;
   CK_ULONG ulPrivateDataLen;
   CK_OBJECT_HANDLE hPrivateData;
   CK_ULONG ulPublicDataLen2;
   CK_BYTE_PTR pPublicData2;
} CK_X9_42_DH2_DERIVE_PARAMS;
```

Поля данной структуры имеют следующие значения:

kdf функция деривации ключа, примененная к совместному секретному значению;
ulOtherInfoLen длина другой информации в байтах;
pOtherInfo некоторые данные, общие для обеих сторон;
ulPublicDataLen длина в байтах первого открытого ключа Диффи – Хеллмана X9.42 другой стороны;

pPublicData указатель на значение первого открытого ключа Диффи – Хеллмана X9.42;
ulPrivateDataLen длина в байтах второго закрытого ключа Диффи – Хеллмана X9.42;
hPrivateData управление ключами для значения второго закрытого ключа Диффи – Хеллмана X9.42;
ulPublicDataLen2 длина в байтах второго открытого ключа Диффи – Хеллмана X9.42 другой стороны;
pPublicData2 указатель на значение второго открытого ключа Диффи – Хеллмана X9.42.

Если применяется функция деривации ключа **CKD_NULL**, значение *pOtherInfo* должно быть NULL, а значение *ulOtherInfoLen* должно быть ноль. Если применяется функция деривации ключа **CKD_ SHA1_KDF_ASN1**, может быть предложено значение *pOtherInfo*, содержащее восьмизначную строку, определенную в стандарте кодирования ASN.1 DER, и состоящее из некоторых обязательных и
необязательных данных, которые обе стороны сделали общими, желая обеспечить одинаковый доступ к общим секретным данным. Если применяется функция деривации ключа **CKD_SHA1_KDF_ CONCATENATE**, может быть предложено необязательное значение *pOtherInfo*, состоящее из некоторых данных, которые обе стороны сделали общими, желая обеспечить одинаковый доступ к общим секретным данным. В противном случае значение *pOtherInfo* должно быть NULL, а значение *ulOtherInfoLen* должно быть ноль.

CK_X9_42_DH2_DERIVE_PARAMS_PTR является указателем на **CK_X9_42_DH2_DERIVE_PARAMS**.

♦ **CK_X9_42_MQV_DERIVE_PARAMS, CK_X9_42_MQV_DERIVE_PARAMS_PTR**

CK_X9_42_MQV_DERIVE_PARAMS представляет собой структуру, предоставляющую параметры для механизмов деривации ключа **CKM_X9_42_MQV_DERIVE**, где каждая сторона выдает две пары ключей. Данная структура задается следующим образом:

```
typedef struct CK_X9_42_MQV_DERIVE_PARAMS {
   CK_X9_42_DH_KDF_TYPE kdf;
   CK_ULONG ulOtherInfoLen;
   CK_BYTE_PTR pOtherInfo;
   CK_ULONG ulPublicDataLen;
   CK_BYTE_PTR pPublicData;
   CK_ULONG ulPrivateDataLen;
   CK_OBJECT_HANDLE hPrivateData;
   CK_ULONG ulPublicDataLen2;
   CK_BYTE_PTR pPublicData2;
   CK_OBJECT_HANDLE publicKey;
} CK_X9_42_MQV_DERIVE_PARAMS;
```

Поля данной структуры имеют следующие значения:

kdf функция деривации ключа, примененная к совместному секретному значению;
ulOtherInfoLen длина другой информации в байтах;
pOtherInfo некоторые данные, общие для обеих сторон;
ulPublicDataLen длина в байтах первого открытого ключа Диффи – Хеллмана X9.42 другой стороны;
pPublicData указатель на значение первого открытого ключа Диффи – Хеллмана X9.42 другой стороны;
ulPrivateDataLen длина в байтах второго открытого ключа Диффи – Хеллмана X9.42;
hPrivateData управление ключами для значения второго секретного ключа Диффи – Хеллмана X9.42;
ulPublicDataLen2 длина в байтах второго открытого ключа Диффи – Хеллмана X9.42 другой стороны;
pPublicData2 указатель на значение второго открытого ключа Диффи – Хеллмана X9.42 другой стороны;
publicKey управление временным открытым ключом первой стороны.

Если применяется функция деривации ключа **CKD_NULL**, значение *pOtherInfo* должно быть NULL, а значение *ulOtherInfoLen* должно быть ноль. Если применяется функция деривации ключа **CKD_ SHA1_KDF_ASN1**, может быть предложено значение *pOtherInfo*, содержащее восьмизначную строку, определенную в стандарте кодирования ASN.1 DER, и состоящее из некоторых обязательных и необязательных данных, которые обе стороны сделали общими, желая обеспечить одинаковый доступ

к общим секретным данным. Если применяется функция деривации ключа **CKD_SHA1_KDF_CONCATENATE**, может

быть предложено необязательное значение *pOtherInfo*, состоящее из некоторых данных, которые обе стороны сделали общими, желая обеспечить одинаковый доступ к общим секретным данным. В противном случае значение *pOtherInfo* должно быть NULL, а значение *ulOtherInfoLen* должно быть ноль.

CK_X9_42_MQV_DERIVE_PARAMS_PTR является указателем на **CK_X9_42_MQV_DERIVE_PARAMS**.

12.4.12 Генерация пары ключей Диффи – Хеллмана X9.42

Механизм генерации пары ключей Диффи – Хеллмана X9.42, обозначенный как **CKM_X9_42_DH_ PKCS_KEY_PAIR_GEN**, представляет собой механизм генерации пары ключей, основанный на согласовании ключей Диффи – Хеллмана, как это определено в ANSI X9.42.

Данный механизм не имеет параметра.

Этот механизм генерирует пары, состоящие из открытого и секретного ключей Диффи – Хеллмана X9.42, отдельно с первичным и базовым значениями, как определено в атрибутах **CKA_PRIME, CKA_BASE** и **CKA_SUBPRIME** шаблона открытого ключа.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому открытому ключу, а также атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_BASE** и **CKA_VALUE** к новому секретному ключу. Другие атрибуты, которые требуются для типов открытого и секретного ключей Диффи – Хеллмана X9.42, должны быть указаны в шаблонах.

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения Диффи – Хеллмана X9.42 в битах.

12.4.13 Генерация параметров домена Диффи – Хеллмана X9.42

Механизм генерации параметра домена Диффи – Хеллмана X9.42, обозначенный как **CKM_X9_42_DH_PARAMETER_GEN**, является механизмом генерации параметра домена, основанным на согласовании ключей Диффи – Хеллмана X9.42, определенном в ANSI X9.42.

Данный механизм не имеет параметра.

Этот механизм генерирует параметры домена Диффи – Хеллмана X9.42 с определенной первичной длиной в битах, как определено в атрибутах **CKA_PRIME_BITS** и **CKA_SUBPRIME_BITS** шаблона для параметров домена.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_BASE, CKA_PRIME_BITS** и **CKA_SUBPRIME_BITS** к новому объекту. В шаблоне могут также быть заданы другие атрибуты, поддерживаемые различными типами параметра домена Диффи-Хеллмана X9.42, либо им могут быть даны начальные значения по умолчанию.

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения Диффи – Хеллмана X9.42 в битах.

12.4.14 Деривация ключей Диффи – Хеллмана X9.42

Механизм деривации ключей Диффи – Хеллмана X9.42, обозначенный как **CKM_X9_42_DH_DERIVE**, представляет собой механизм деривации ключей, основанный на согласовании ключей Диффи – Хеллмана, как это определено в ANSI X9.42, когда каждая из сторон представляет одну пару ключей, при чем обе стороны используют одинаковые параметры домена Диффи – Хеллмана X9.42.

Данный механизм имеет параметр структуры **CK_X9_42_DH1_DERIVE_PARAMS**.

Данный механизм получает секретное значение, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне. Отметим, что для подтверждения работоспособности данного механизма может потребоваться использование атри-бута **CKA_VALUE** в качестве ключа механизма общей длины MAC (например, **CKM_SHA_1_HMAC_ GENERAL**) для использования с некоторыми проверочными данными.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE,

тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.

- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон первичных размеров атрибута **CKA_PRIME** в соответствии со стандартом Диффи – Хеллмана X9.42.

12.4.15 Гибридная деривация ключей Диффи – Хеллмана X9.42

Механизм гибридной деривации ключей Диффи – Хеллмана X9.42, обозначенный как **CKM_X9_42_DH_HYBRID_DERIVE**, представляет собой механизм деривации ключей, основанный на схеме гибридного согласования ключей Диффи – Хеллмана, как это определено в ANSI X9.42, когда каждая из сторон представляет две пары ключей, причем обе стороны используют одинаковые параметры домена Диффи – Хеллмана X9.42.

Данный механизм имеет параметр структуры **CK_X9_42_DH2_DERIVE_PARAMS**.

Данный механизм получает секретное значение, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне. Отметим, что для подтверждения работоспособности данного механизма может потребоваться использование атрибута **CKA_VALUE** в качесте ключа механизма общей длины MAC (например, **CKM_SHA_1_HMAC_GENERAL**) для использования с некоторыми проверочными данными.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, как его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон первичных размеров атрибута **CKA_PRIME** в соответствии со стандартом Диффи – Хеллмана X9.42.

12.4.16 Деривация ключей Диффи – Хеллмана – Менезеса – Ку – Вэнстоуна X9.42

Механизм деривации ключей Диффи – Хеллмана – Менезеса – Ку – Вэнстоуна (MQV) X9.42, обозначенный как **CKM_X9_42_MQV_DERIVE**, представляет собой механизм деривации ключей, основанный на схеме MQV, как определено в ANSI X9.42, причем обе стороны используют одинаковые параметры домена Диффи – Хеллмана X9.42.

Данный механизм имеет параметр структуры **CK_X9_42_MQV_DERIVE_PARAMS**.

Данный механизм получает секретное значение, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне. Отметим, что для подтверждения работоспособности данного механизма может потребоваться использование атри-бута **CKA_VALUE** в качестве ключа механизма общей длины MAC (например, **CKM_SHA_1_HMAC_ GENERAL**) для использования с некоторыми проверочными данными.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон первичных размеров атрибута **CKA_PRIME** в соответствии со стандартом Диффи – Хеллмана X9.42.

12.5 KEA

12.5.1 Определения

В этом разделе тип ключа CKK_KEA определяется как CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_KEA_KEY_PAIR_GEN
CKM_KEA_KEY_DERIVE
```

12.5.2 Параметры механизма KEA

♦ **CK_KEA_DERIVE_PARAMS; CK_KEA_DERIVE_PARAMS_PTR**

CK_KEA_DERIVE_PARAMS – структура, которая обеспечивает параметрами механизм **CKM_KEA_DERIVE**. Данная структура задается следующим образом:

```
typedef struct CK_KEA_DERIVE_PARAMS {
  CK_BBOOL isSender;
  CK_ULONG ulRandomLen;
  CK_BYTE_PTR pRandomA;
  CK_BYTE_PTR pRandomB;
  CK_ULONG ulPublicDataLen;
  CK_BYTE_PTR pPublicData;
} CK_KEA_DERIVE_PARAMS;
```

Поля данной структуры имеют следующие значения:

isSender вариант для генерирования ключа, именуемого TEK. Данное значение оценивается как CK_TRUE, если отправитель (создатель) генерирует TEK, и как CK_FALSE, если получатель вновь генерирует TEK;
ulRandomLen размер случайных величин Ra и Rb в байтах;
pRandomA указатель на данные Ra;
pRandomB указатель на данные Rb;
ulPublicDataLen размер открытого ключа KEA другой стороны;
pPublicData указатель на значение открытого ключа KEA другой стороны.

CK_KEA_DERIVE_PARAMS_PTR является указателем на **CK_KEA_DERIVE_PARAMS**.

12.5.3 Объекты открытого ключа KEA

Объекты открытого ключа KEA (класс объекта – **CKO_PUBLIC_KEY,** тип ключа – **CKK_KEA**) содержат открытые ключи KEA. В таблице 67 определяются атрибуты открытых ключей KEA в дополнение к общим атрибутам, заданным для объектов данного класса.

Таблица 67 – Атрибуты объекта открытого ключа KEA

Атрибут	Тип данных	Значение
CKA_PRIME [1, 3]	Большое целое	Первичное *p* (от 512 до 1024 бит, шагами по 64 бит)
CKA_SUBPRIME [1, 3]	Большое целое	Предпервичное *q* (160 бит)
CKA_BASE [1, 3]	Большое целое	Базовое *g* (от 512 до 1024 бит, шагами по 64 бит)

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4]	Большое целое	Открытое значение *y*
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME**, **CKA_SUBPRIME** и **CKA_BASE** совместно составляют «параметры домена KEA».

Ниже приводится образец шаблона для создания объекта открытого ключа KEA:

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_KEA;
CK_UTF8CHAR label[] = "A KEA public key object";
CK_BYTE prime[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_PRIME, prime, sizeof(prime)},
  {CKA_SUBPRIME, subprime, sizeof(subprime)},
  {CKA_BASE, base, sizeof(base)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.5.4 Объекты секретного ключа KEA

Объекты секретного ключа KEA (класс объекта – **CKO_PRIVATE_KEY,** тип ключа – **CKK_KEA**) содержат секретные ключи KEA. В таблице 68 определены атрибуты секретных ключей KEA в дополнение к общим атрибутам, заданным для объектов данного класса.

Таблица 68 – Атрибуты секретного ключа KEA

Атрибут	Тип данных	Значение
CKA_PRIME [1, 4, 6]	Большое целое	Первичное *p* (от 512 до 1024 бит с шагом по 64 бит)
CKA_SUBPRIME [1, 4, 6]	Большое целое	Предпервичное *q* (160 бит)
CKA_BASE [1, 4, 6]	Большое целое	Базовое *g* (от 512 до 1024 бит с шагом по 64 бит)
CKA_VALUE [1, 4, 6, 7]	Большое целое	Секретное значение *x*
См. сноски в таблице 15.		

Значения атрибутов **CKA_PRIME**, **CKA_SUBPRIME** и **CKA_BASE** совместно составляют «параметры домена KEA».

Следует отметить, что при генерации секретного ключа KEA параметры KEA *не* задаются в шаблоне данного ключа. Причина состоит в том, что секретные ключи KEA генерируются только как часть *пары* ключей KEA, а параметры домена KEA для этой пары определены в шаблоне для открытого ключа KEA.

Ниже представлен образец шаблона для создания объекта секретного ключа KEA:

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_KEA;
CK_UTF8CHAR label[] = "Объект секретного ключа KEA";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE prime[] = {...};
CK_BYTE subprime[] = {...};
CK_BYTE base[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
```

```
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_SUBJECT, subject, sizeof(subject)},
    {CKA_ID, id, sizeof(id)},
    {CKA_SENSITIVE, &true, sizeof(true)},
    {CKA_DERIVE, &true, sizeof(true)},
    {CKA_PRIME, prime, sizeof(prime)},
    {CKA_SUBPRIME, subprime, sizeof(subprime)},
    {CKA_BASE, base, sizeof(base)},
    {CKA_VALUE, value, sizeof(value)}
};
```

12.5.5 Генерация пары ключей KEA

Механизм генерации пары ключей KEA, обозначенный как **CKM_KEA_KEY_PAIR_GEN**, генерирует пары ключей для алгоритма обмена ключами (Key Exchange Algorithm, KEA), как это определено в распространенном NIST документе «SKIPJACK and KEA Algorithm Specification Version 2.0» от 29 мая 1998 года.

Данный механизм не имеет параметра.

Этот механизм генерирует пары, состоящие из открытого и секретного ключей KEA, отдельно с первичным, предпервичным и базовым значениями, как определено в атрибутах **CKA_PRIME, CKA_ SUBPRIME** и **CKA_BASE** шаблона открытого ключа. Следует отметить, что данная версия системы Cryptoki не содержит механизма для генерации этих параметров домена KEA.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому открытому ключу, а также атрибуты **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_PRIME**, **CKA_BASE** и **CKA_ VALUE** к новому секретному ключу. Другие атрибуты, которые требуются для типов открытого и секретного ключей KEA (в частности, флаги, указывающие, какие функции поддерживают данные ключи), могут быть также указаны в шаблонах, либо им могут быть присвоены первичные значения по умолчанию.

Для этого механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров первичного значения Диффи – Хеллмана в битах.

12.5.6 Деривация ключа KEA

Механизм деривации ключей KEA, обозначенный как **CKM_KEA_DERIVE**, представляет собой механизм деривации ключей, основанный на алгоритме обмена ключами (Key Exchange Algorithm, KEA), как это определено в распространенном NIST документе «SKIPJACK and KEA Algorithm Specification Version 2.0» от 29 мая 1998 года.

Данный механизм имеет параметр структуры **CK_KEA_DERIVE_PARAMS**.

Этот механизм получает секретный ключ, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Как определено в требованиях, алгоритм KEA может применяться в двух различных операционных режимах: в полном режиме и в режиме электронной почты. Полный режим представляет собой двухэтапную последовательность деривации ключа, требующую от обеих сторон обмена параметрами в реальном времени. Режим электронной почты представляет собой одноэтапную последовательность деривации ключа, не требующую обмена параметрами в реальном времени. Согласно договоренности режим электронной почты предусматривает использование в качестве предусмотренного в KEA параметра R_b (*pRandomB* – случайная величина *B*) фиксированного значения (1).

Как детально изложено в приведенной ниже таблице, работа данного механизма зависит от двух значений в прилагаемой структуре **CK_KEA_DERIVE_PARAMS.** Следует отметить, что во всех случаях буферы данных, на которые указывают поля *pRandomA* и *pRandomB* структуры параметра, должны быть назначены абонентом прежде, чем будет активирован механизм **C_DeriveKey**. Кроме того, значения, на которые указывают поля *pRandomA* и *pRandomB*, представлены как данные Cryptoki «Большое целое» (т. е. последовательность байт, где первым идет наиболее старший по разрядности).

Таблица 69 – Значения и операции параметра KEA

Булево значение *isSender*	Значение большого целого *pRandomB*	Действие носителя (после проверки значений параметра и шаблона)
CK_TRUE	0	Вычислить значение KEA R_a, поместить его в *pRandomA*, выдать сообщение CKR_OK. Объект полученного ключа не создается
CK_TRUE	1	Вычислить значение KEA R_a, поместить его в *pRandomA*, получить значение ключа, используя режим электронной почты, создать объект ключа, выдать сообщение CKR_OK
CK_TRUE	> 1	Вычислить значение KEA R_a, поместить его в *pRandomA*, получить значение ключа, используя полный режим, создать объект ключа, выдать сообщение CKR_OK
CK_FALSE	0	Вычислить значение KEA R_b, поместить его в *pRandomB*, выдать сообщение CKR_OK. Объект полученного ключа не создается
CK_FALSE	1	Получить значение ключа, используя режим электронной почты, создать объект ключа, выдать сообщение CKR_OK
CK_FALSE	> 1	Получить значение, используя полный режим, создать объект ключа, выдать сообщение CKR_OK

Следует отметить, что значение параметра *pRandomB* = 0 является флагом, указывающим, что механизм KEA активирован для вычисления открытого случайного значения данной стороны (R_a или R_b, для отправителя или получателя соответственно), а не для деривации ключа. В этих случаях любой шаблон объекта, поданный как аргумент **C_DeriveKey** *pTemplate*, должен быт проигнорирован.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа [2]:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон первичных размеров атрибута **CKA_PRIME** в соответствии со стандартом Диффи – Хеллмана X9.42.

12.6 Упаковка/распаковка секретных ключей

Версии Cryptoki 2.01 и выше позволяют использовать секретные ключи для упаковки и распаковки секретных ключей RSA, секретных ключей Диффи – Хеллмана, секретных ключей Диффи – Хеллмана X9.42, секретных ключей EC (это также относится к ECDSA) и секретных ключей DSA. Для упаковки используется секретный ключ, подвергнутый BER-кодированию в соответствии с PKCS #8 Private-KeyInfo типа ASN.1. В PKCS #8 требуется идентификатор алгоритма для типа секретного ключа. Применяются следующие идентификаторы объекта для требуемых идентификаторов алгоритма:

```
rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }

dhKeyAgreement OBJECT IDENTIFIER ::= { pkcs-3 1 }
```

[2] Отметим, что правила, касающиеся атрибутов **CKA_SENSITIVE**, **CKA_EXTRACTABLE**, **CKA_ALWAYS_SENSITIVE** и **CKA_NEVER_EXTRACTABLE** в версии 2.11, были изменены с целью приведения их в соответствие с политикой, применяемой в отношении других механизмов деривации ключей, таких как **CKM_SSL3_MASTER_KEY_DERIVE**.

```
   dhpublicnumber OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) ansi-
x942(10046) number-type(2) 1 }

   id-ecPublicKey OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) ansi-
x9-62(10045) publicKeyType(2) 1 }

   id-dsa OBJECT IDENTIFIER ::= {
     iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) 1 }
```

где

```
pkcs-1 OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 1 }

pkcs-3 OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 3 }
```

Эти параметры для идентификаторов алгоритма имеют соответственно следующие типы:

```
NULL

DHParameter ::= SEQUENCE {
  prime                   INTEGER, -- p
  base                    INTEGER, -- g
  privateValueLength INTEGER OPTIONAL
}

DomainParameters ::= SEQUENCE {
  prime                   INTEGER, -- p
  base                    INTEGER, -- g
  subprime                INTEGER, -- q
  cofactor                INTEGER OPTIONAL, -- j
  validationParms    ValidationParms OPTIONAL
}

ValidationParms ::= SEQUENCE {
  Seed               BIT STRING, -- начальное число
  PGenCounter        INTEGER – проверка параметра
}

Parameters ::= CHOICE {
  ecParameters ECParameters,
  namedCurve         CURVES.&id({CurveNames}),
  implicitlyCA NULL
}

Dss-Parms ::= SEQUENCE {
  p INTEGER,
  q INTEGER,
  g INTEGER
}
```

Для параметров домена Диффи – Хеллмана X9.42 не следует использовать необязательные поля **cofactor** и **validationParms** при упаковке и распаковке секретных ключей Диффи – Хеллмана X9.42, так как их значения не хранятся внутри носителя.

Для параметров домена EC рекомендуется использовать **namedCurve** при варианте **ecParameters**. Вариант **implicitlyCA** в Cryptoki использоваться не должен.

Для типа PrivateKeyInfo:

- Секретные ключи RSA подвергнуты BER-кодированию в соответствии с требованиями PKCS #1 для типа RSAPrivateKey ASN.1. Данный тип требует, чтобы были представлены значения для *всех* атрибутов, характерных для объектов секретного ключа RSA системы Cryptoki. Иными словами, если в библиотеке Cryptoki не имеется таких значений секретного ключа RSA, как **CKA_MODULUS**,

CKA_PUBLIC_EXPONENT, **CKA_PRIVATE_EXPONENT**, **CKA_PRIME_1**, **CKA_PRIME_2**, **CKA_EXPONENT_1**, **CKA_EXPONENT2** и **CKA_COEFFICIENT**, система не сможет произвести BER-кодирование ключевой величины RSAPrivateKey, из-за чего ключ не смогут подготовить к упаковке.

- Секретные ключи Диффи – Хеллмана представляются как INTEGER, подвергнутые BER-коди-рованию, тип ASN.1.
- Секретные ключи Диффи – Хеллмана X9.42 представляются как INTEGER, подвергнутые BER-кодированию, тип ASN.1.
- Секретные ключи EC (относящиеся также к ECDSA) подвергаются BER-кодированию в соответствии с SECG SEC 1 ECPrivateKey, тип ASN.1:

```
ECPrivateKey ::= SEQUENCE {
  Version            INTEGER { ecPrivkeyVer1(1) } (ecPrivkeyVer1),
  privateKey         OCTET STRING,
  parameters         [0] Parameters OPTIONAL,
  publicKey          [1] BIT STRING OPTIONAL
}
```

Поскольку параметры домена EC помещаются в принадлежащее PKCS #8 поле privateKeyAlgorithm, необязательное поле **parameters** в ECPrivateKey должно быть опущено. Приложение Cryptoki должно иметь возможность распаковать ECPrivateKey, где имеется необязательное поле **publicKey**; однако операции, осуществляемые с этим полем **publicKey,** лежат за пределами функциональности Cryptoki.

- Секретные ключи DSA представляются как INTEGER, подвергнутые BER-кодированию, тип ASN.1.

Если секретный ключ был подвергнут BER-кодированию как величина типа PrivateKeyInfo, результирующая байтовая строка зашифровывается секретным ключом. Это шифрование должно осуществляться в режиме CBC с заполнением PKCS.

Распаковка запакованного секретного ключа производится посредством осуществления процедур, отменяющих результат применения вышеупомянутых процедур. Шифртекст, зашифрованный посредством CBC, расшифровывается, а заполение PKCS удаляется. Данные, полученные таким образом, раскладываются как величина типа PrivateKeyInfo, в результате чего получается ключ, который был упакован. Ошибка может возникнуть, если оригинальный упакованный ключ не был расшифрован должным образом, или если расшифрованные данные с удаленным заполнением не разложены должным образом, или если тип этих данных не совпадает с типом ключа, определенным в шаблоне для нового ключа. Механизм распаковки добавляет к вновь распакованному ключу лишь те атрибуты, которые были заданы для нового ключа в величине типа PrivateKeyInfo; другие атребуты должны быть заданы в шаблоне либо должны принять свои значения по умолчанию.

Более ранние варианты PKCS #11 – Version 2.0 и Version 2.01 использовали идентификатор объекта

```
DSA OBJECT IDENTIFIER ::= { algorithm 12 }
algorithm OBJECT IDENTIFIER ::= {
  iso(1) identifier-organization(3) oiw(14) secsig(3) algorithm(2) }
```

с ассоциированными параметрами

```
DSAParameters ::= SEQUENCE {
  prime1 INTEGER, -- модуль p
  prime2 INTEGER, -- модуль q
  base INTEGER -- базовое g
}
```

для распаковки секретных ключей DSA. Следует отметить, что, хотя обе структуры, применяемые для размещения параметров домена DSA, оказываются идентичными, когда результаты отдельных этапов их преобразований закодированы, два соответствующих идентификатора объекта различаются.

12.7 Общий секретный ключ

12.7.1 Определения

В этом разделе тип ключа «CKK_GENERIC_SECRET» определяется как CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_GENERIC_SECRET_KEY_GEN
```

12.7.2 Объекты общего секретного ключа

Объекты общего секретного ключа (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_GENERIC_ SECRET**) содержат общие секретные ключи. Эти ключи не поддерживают зашифровывание, расшифровку, постановку подписи или верификацию; напротив, из них могут быть получены другие ключи. В таблице 70 определены атрибуты объектов общего секретного ключа в дополнение к общим атри-бутам, заданным для объектов данного класса.

Данные типы ключей используются в различных механизмах, описанных в настоящем разделе.

Таблица 70 – Атрибуты объекта общего секретного ключа

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (произвольная длина)
CKA_VALUE_LEN [2, 3]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Ниже приведен образец шаблона для создания объекта общего секретного ключа:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_GENERIC_SECRET;
CK_UTF8CHAR label[] = “A generic secret key object”;
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_DERIVE, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

CKA_CHECK_VALUE: Значение данного атрибута получено из объекта ключа посредством взятия первых трех байт из результата хэширования SHA-1 атрибута CKA_VALUE объекта общего секретного ключа.

12.7.3 Генерация общего секретного ключа

Механизм генерации общего секретного ключа, обозначенный как **CKM_GENERIC_SECRET_ KEY_GEN**, применяется для генерации общих секретных ключей. Сгенерированные ключи принимают любые атрибуты, предоставленные в шаблоне, переданном на запрос **C_GenerateKey**, а длина ключа, который предстоит сгенерировать, задается атрибутом **CKA_VALUE_LEN**.

Данный механизм не имеет параметра.

Представленный шаблон должен задавать значение для атрибута **CKA_VALUE_LEN**. Если шаблон задает тип и класс объекта, они должны иметь следующие значения:

CK_OBJECT_CLASS = CKO_SECRET_KEY;
CK_KEY_TYPE = CKK_GENERIC_SECRET;

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа в битах.

12.8 Механизмы HMAC

Описание алгоритма HMAC имеется в документах **RFC2104** и **FIPS 198**. Секретный ключ HMAC должен соответствовать типу общего секретного ключа PKCS11. Такие ключи, предназначенные для использования с операциями HMAC, могут создаваться с применением C_CreateObject или C_GenerateKey.

RFC задает также проверочные векторы для различных механизмов HMAC, основанных на хэш-функциях. Описание этих механизмов имеется в описаниях механизмов соответствующих хэш-функций. Для соблюдения соответствия указанным проверочным векторам, необходимо консультироваться с RFC.

12.9 RC2

RC2 представляет собой блочный шифр, авторские права на который принадлежат RSA Security. Он располагает ключами различной длины и дополнительным параметром, именуемым «эффективное число бит в поисковом пространстве RC2», который может принимать значения в диапазоне 1 – 1024, включая границы диапазона. Это эффективное число бит в поисковом пространстве RC2 иногда задается «номером версии» RC2; однако этот «номер версии» *не является* тем же самым, что «эффективное число бит». Существует установленный порядок конверсии одного в другое.

12.9.1 Определения

В этом разделе тип ключа «CKK_RC2» определяется как CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_RC2_KEY_GEN
CKM_RC2_ECB
CKM_RC2_CBC
CKM_RC2_MAC
CKM_RC2_MAC_GENERAL
CKM_RC2_CBC_PAD
```

12.9.2 Объекты секретного ключа RC2

Объекты секретного ключа RC2 (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_RC2**) содержат ключи RC2. В приведенной ниже таблице определяются атрибуты объекта секретного ключа RC2 в дополнение к общим атрибутам, заданным для данного класса объектов:

Таблица 71 – Атрибуты объекта секретного ключа RC2

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (от 1 до 128 байт)
CKA_VALUE_LEN [2, 3]	CK_ULONG	Длина в байтах значения ключа
См. сноски в таблице 15.		

Ниже приведен образец шаблона для создания объекта секретного ключа RC2:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_RC2;
CK_UTF8CHAR label[] = "Объект секретного ключа RC2";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.9.3 Параметры механизма RC2

- **CK_RC2_PARAMS; CK_RC2_PARAMS_PTR**

CK_RC2_PARAMS предоставляет параметры для механизмов **CKM_RC2_ECB** и **CKM_RC2_MAC**. Данный параметр содержит эффективное число бит в поисковом пространстве RC2. Он задается следующим образом:

```
typedef CK_ULONG CK_RC2_PARAMS;
```

CK_RC2_PARAMS_PTR является указателем на **CK_RC2_PARAMS**.

♦ **CK_RC2_CBC_PARAMS; CK_RC2_CBC_PARAMS_PTR**

CK_RC2_CBC_PARAMS является структурой, предоставляющей параметры механизмам **CKM_RC2_CBC** и **CKM_RC2_CBC_PAD**. Она задается следующим образом:

```
typedef struct CK_RC2_CBC_PARAMS {
  CK_ULONG ulEffectiveBits;
  CK_BYTE iv[8];
} CK_RC2_CBC_PARAMS;
```

Поля данной структуры имеют следующие значения:

ulEffectiveBits эффективное число бит в поисковом пространстве RC2;

iv вектор инициализации (initialization vector, IV) для режима цепочки шифрблоков.

CK_RC2_CBC_PARAMS_PTR является указателем на **CK_RC2_CBC_PARAMS**.

♦ **CK_RC2_MAC_GENERAL_PARAMS; CK_RC2_MAC_GENERAL_PARAMS_PTR**

CK_RC2_MAC_GENERAL_PARAMS является структурой, предоставляющей параметры механизму **CKM_RC2_MAC_GENERAL**. Она задается следующим образом:

```
typedef struct CK_RC2_MAC_GENERAL_PARAMS {
  CK_ULONG ulEffectiveBits;
  CK_ULONG ulMacLength;
} CK_RC2_MAC_GENERAL_PARAMS;
```

Поля данной структуры имеют следующие значения:

ulEffectiveBits эффективное число бит в поисковом пространстве RC2;

ulMacLength длина результирующего MAC в байтах.

CK_RC2_MAC_GENERAL_PARAMS_PTR является указателем на **CK_RC2_MAC_GENERAL_PARAMS**.

12.9.4 Генерация ключа RC2

Механизм генерации ключа RC2, обозначенный как **CKM_RC2_KEY_GEN**, является механизмом генерации ключа для блочного шифра RC2 компании RSA Security.

Данный механизм не имеет параметра.

Этот механизм генерирует ключи RC2 определенной длины в байтах, как определено в атрибуте **CKA_VALUE_LEN** шаблона данного ключа.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому ключу. Остальные атрибуты, поддерживаемые ключом типа RC2 (особенно флаги, указывающие, какие функции данный ключ поддерживает), могут быть заданы в шаблоне, или им могут быть присвоены начальные значения по умолчанию.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC2 в битах.

12.9.5 RC2-ECB

Механизм RC2-ECB, обозначенный как **CKM_RC2_ECB**, является механизмом для одностороннего и многостороннего шифрования и расшифровки, упаковки ключей и распаковки ключей, основанным на блочном шифре компании RSA Security и на режиме электронной кодовой книги, как определено в FIPS PUB 81.

Данный механизм имеет параметр **CK_RC2_PARAMS**, который указывает эффективное число бит в поисковом пространстве RC2.

Данный механизм может упаковывать и распаковывать любой секретный ключ. Конкретный носитель не всегда может упаковывать/распаковывать любой поддерживаемый им секретный ключ. Для упаковки этот механизм зашифровывает значение атрибута **CKA_VALUE** упаковываемого ключа, добавляя на замыкающем конце до семи нулевых байт, чтобы результирующая длина была кратна восьми. Данные вывода имеют ту же длину, что и дополненные данные ввода. Этот механизм не упаковывает тип ключа или любую иную информацию о ключе, за исключением длины ключа; приложение должно производить это отдельно.

Для распаковки данный механизм расшифровывает упакованный ключ, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы

в шаблоне.

Ограничения на типы ключей и на длину данных приведены в таблице 72.

Таблица 72 – RC2-ECB: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	RC2	Кратна 8	Та же, что и длина ввода	Без заключительной части
C_Decrypt	RC2	Кратна 8	Та же, что и длина ввода	Без заключительной части
C_WrapKey	RC2	Любая	Длина ввода, округленная до числа, кратного 8	
C_UnwrapKey	RC2	Кратна 8	Определяется типом распаковываемого ключа или CKA_VALUE_LEN	

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон эффективного числа бит ключа RC2.

12.9.6 RC2-CBC

Механизм RC2-CBC, обозначенный как **CKM_RC2_CBC**, является механизмом, предназначенным для одностороннего и многостороннего шифрования и расшифровки, упаковки ключей и распаковки ключей, основанным на блочном шифре RC2 компании RSA Security и на режиме цепочки шифр-блоков, как определено в FIPS PUB 81.

Данный механизм имеет параметр структуры **CK_RC2_CBC_PARAMS**, где первое поле указывает эффективное число бит в поисковом пространстве RC2, а следующее поле является вектором инициализации для режима цепочки шифрблоков.

Данный механизм может упаковывать и распаковывать любой секретный ключ. Конкретный носитель не всегда может упаковывать/распаковывать любой поддерживаемый им секретный ключ. Для упаковки этот механизм зашифровывает значение атрибута **CKA_VALUE** упаковываемого ключа, добавляя на замыкающем конце до семи нулевых байт, чтобы результирующая длина была кратна восьми. Данные вывода имеют ту же длину, что и дополненные данные ввода. Этот механизм не упаковывает тип ключа или любую иную информацию о ключе, за исключением длины ключа; приложение должно производить это отдельно.

Для распаковки данный механизм расшифровывает упакованный ключ, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона (если он у него есть и если данный тип ключа его поддерживает) и в соответствии с атрибутом **CKA_VALUE_LEN** шаблона (при усечении удаляются байты с переднего конца секретного значения). Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Ограничения на типы ключей и на длину данных приведены в таблице 73.

Таблица 73 – RC2-CBC: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	RC2	Кратна 8	Та же, что и длина ввода	Без заключительной части
C_Decrypt	RC2	Кратна 8	Та же, что и длина ввода	Без заключительной части
C_WrapKey	RC2	Любая	Длина ввода, округленная до числа, кратного 8	
C_UnwrapKey	RC2	Кратна 8	Определяется типом распаковываемого ключа или CKA_VALUE_LEN	

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон эффективного числа бит ключа RC2.

12.9.7 RC2-CBC с заполнением PKCS

Механизм RC2-CBC с заполнением PKCS, обозначенный **CKM_RC2_CBC_PAD**, является механизмом для одностороннего и многостороннего шифрования и расшифровки, упаковки ключей и распаковки ключей, основанным на блочном шифре RC2 компании RSA Security, на режиме цепочки шифрблоков, как определено в FIPS PUB 81, а также на методе заполнения шифрблоками, описанном в PKCS #7.

Данный механизм имеет параметр структуры **CK_RC2_CBC_PARAMS**, где первое поле указывает эффективное число бит в поисковом пространстве RC2, а следующее поле является вектором инициализации.

Имеющееся в этом механизме заполнение PKCS позволяет извлекать значение длины открытого текста из значения шифртекста. Таким образом, при распаковке ключей при помощи этого механизма, не нужно задавать значение для атрибута **CKA_VALUE_LEN**.

В дополнение к возможности упаковки и распаковки секретных ключей этот механизм может упаковывать и распаковывать секретные ключи RSA, Диффи – Хеллмана, Диффи – Хеллмана X9.42, EC (относящиеся также к ECDSA) и DSA (подробнее см. 12.6). Данные в таблице 74 по ограничениям на длину данных при упаковке и распаковке ключей не относятся к упаковке и распаковке секретных ключей.

Ограничения на типы ключа и длину данных приведены в таблице 74.

Таблица 74 – RC2-CBC с заполнением PKCS: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Encrypt	RC2	Любая	Длина ввода, округленная до числа, кратного 8
C_Decrypt	RC2	Кратная 8	Короче длины ввода на величину от 1 до 8 байт
C_WrapKey	RC2	Любая	Длина ввода, округленная до числа, кратного 8
C_UnwrapKey	RC2	Кратная 8	Короче длины ввода на величину от 1 до 8 байт

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон эффективного числа бит ключа RC2.

12.9.8 Механизм общей длины RC2-MAC

Механизм общей длины RC2-MAC, обозначенный как **CKM_RC2_MAC_GENERAL**, является механизмом для односторонних и многосторонних процедур постановки подписи и верификации, основанным на блочном шифровании RC2 компании RSA Security и на аутентификации данных, как определено в FIPS PUB 113.

Данный механизм имеет параметр структуры **CK_RC2_MAC_GENERAL_PARAMS**, который задает эффективное число бит в поисковом пространстве RC2 и длину вывода, желательную для данного механизма.

Байты вывода этого механизма берутся из начальной части финального шифрблока RC2, полученного посредством процесса MAC.

Ограничения на типы ключа и длину данных приведены в таблице 75.

Таблица 75 – Механизм общей длины RC2-MAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	RC2	Любая	0 – 8, как указано в параметрах
C_Verify	RC2	Любая	0 – 8, как указано в параметрах

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон эффективного числа бит ключа RC2.

12.9.9 RC2-MAC

Механизм RC2-MAC, обозначенный как **CKM_RC2_MAC**, является частным случаем механизма общей длины RC2-MAC (см. 12.9.8). Вместо использования параметра **CK_RC2_MAC_GENERAL_PARAMS** он использует параметр **CK_RC2_PARAMS**, в котором содержится только эффективное число бит поискового пространства RC2. RC2-MAC всегда производит и верифицирует 4-байтные MAC.

Ограничения на типы ключа и длину данных приведены в таблице 76.

Таблица 76 – RC2-MAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	RC2	Любая	4
C_Verify	RC2	Любая	4

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон эффективного числа бит ключа RC2.

12.10 RC4

12.10.1 Определения

В этом разделе тип ключа «CKK_RC4» определяется как CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:
CKM_RC4_KEY_GEN
CKM_RC4

12.10.2 Объекты секретного ключа RC4

Объекты секретного ключа RC4 (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_RC4**) содержат ключи RC4. В таблице 77 определены атрибуты объекта секретного ключа RC4 в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 77 – Объект секретного ключа RC4

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (от 1 до 256 байт)
CKA_VALUE_LEN [2, 3, 6]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Далее приводится образец шаблона для создания объекта секретного ключа RC4:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_RC4;
CK_UTF8CHAR label[] = "Объект секретного ключа RC4";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.10.3 Генерация ключа RC4

Механизм генерации ключа RC4, обозначенный как **CKM_RC4_KEY_GEN**, представляет собой механизм генерации ключа для поточного шифра RC4, являющегося собственностью компании RSA Security.

Данный механизм не имеет параметра.

Этот механизм генерирует ключи RC4 определенной длины в байтах, как определено в атрибуте **CKA_VALUE_LEN** шаблона для данного ключа.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому ключу. Остальные атрибуты, поддерживаемые ключом типа RC4 (особенно флаги, указывающие, какие функции данный ключ поддерживает), могут быть заданы в шаблоне, или им могут быть присвоены
начальные значения по умолчанию.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC4 в битах.

12.10.4 Механизм RC4

Механизм RC4, обозначенный как **CKM_RC4**, является механизмом для одностороннего и многостороннего шифрования и расшифровки на основе поточного шифра RC4, являющегося собственностью компании RSA Security.

Этот механизм не имеет параметра.

Ограничения на типы ключа и длину данных ввода и вывода приведены в таблице 78.

Таблица 78 – RC4: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	RC4	Любая	Та же, что и длина ввода	Без заключительной части
C_Decrypt	RC4	Любая	Та же, что и длина ввода	Без заключительной части

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC4 в битах.

12.11 RC5

RC5 представляет собой блочный шифр с изменяемыми параметрами, запатентованный компанией RSA Security. У него изменяемая величина слова (wordsize), изменяемый размер ключа и изменяемое количество этапов. Величина блока (blocksize) у RC5 всегда идентична двойной величине слова (wordsize).

12.11.1 Определения

В этом разделе тип ключа «CKK_ RC5» определяется как CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_RC5_KEY_GEN
CKM_RC5_ECB
CKM_RC5_CBC
CKM_RC5_MAC
CKM_RC5_MAC_GENERAL
CKM_RC5_CBC_PAD
```

12.11.2 Объекты секретного ключа RC5

Объекты секретного ключа RC5 (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_RC5**) содержат ключи RC5. В таблице 79 определены атрибуты объектов секретного ключа RC5 в дополнение к общим атрибутам, определенным для объектов этого класса.

Таблица 79 – Объект секретного ключа RC5

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (от 0 до 255 байт)
CKA_VALUE_LEN [2, 3, 6]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Ниже приводится образец шаблона для создания объекта секретного ключа RC5:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_RC5;
CK_UTF8CHAR label[] = "Объект секретного ключа RC5";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.11.3 Параметры механизма RC5

♦ **CK_RC5_PARAMS; CK_RC5_PARAMS_PTR**

CK_RC5_PARAMS предоставляет параметры для механзмов **CKM_RC5_ECB** и **CKM_RC5_MAC**. Он задается следующим образом:

```
typedef struct CK_RC5_PARAMS {
```

```
  CK_ULONG ulWordsize;
  CK_ULONG ulRounds;
} CK_RC5_PARAMS;
```

Поля данной структуры имеют следующие значения:
ulWordsize размер слова (wordsize) шифра RC5 в байтах;
ulRounds число этапов шифрования RC5.
CK_RC5_PARAMS_PTR является указателем на **CK_RC5_PARAMS**.

♦ **CK_RC5_CBC_PARAMS; CK_RC5_CBC_PARAMS_PTR**

CK_RC5_CBC_PARAMS является структурой, предоставляющей параметры механизмам **CKM_RC5_CBC** и **CKM_RC5_CBC_PAD**. Она задается следующим образом:

```
typedef struct CK_RC5_CBC_PARAMS {
  CK_ULONG ulWordsize;
  CK_ULONG ulRounds;
  CK_BYTE_PTR pIv;
  CK_ULONG ulIvLen;
} CK_RC5_CBC_PARAMS;
```

Поля данной структуры имеют следующие значения:
ulWordsize размер слова (wordsize) шифра RC5 в байтах;
ulRounds число этапов шифрования RC5;
pIv указатель на вектор инициализации (initialization vector, IV) для шифрования CBC;
ulIvLen длина вектора инициализации (должна быть та же, что и размер блока).
CK_RC5_CBC_PARAMS_PTR является указателем на **CK_RC5_CBC_PARAMS**.

♦ **CK_RC5_MAC_GENERAL_PARAMS; CK_RC5_MAC_GENERAL_PARAMS_PTR**

CK_RC5_MAC_GENERAL_PARAMS является структурой, которая предоставляет параметры механизму **CKM_RC5_MAC_GENERAL.** Она задается следующим образом:

```
typedef struct CK_RC5_MAC_GENERAL_PARAMS {
  CK_ULONG ulWordsize;
  CK_ULONG ulRounds;
  CK_ULONG ulMacLength;
} CK_RC5_MAC_GENERAL_PARAMS;
```

Поля данннной структуры имеют следующие значения:
ulWordsize размер слова (wordsize) шифра RC5 в байтах;
ulRounds число этапов шифрования RC5;
ulMacLength длина результирующего MAC в байтах.
CK_RC5_MAC_GENERAL_PARAMS_PTR является указателем на **CK_RC5_MAC_GENERAL_PARAMS**.

12.11.4 Генерация ключа RC5

Механизм генерации ключа RC5, обозначенный как **CKM_RC5_KEY_GEN**, является механизмом генерации ключа для блочного шифра RC5 компании RSA Security.

Данный механизм не имеет параметра.

Этот механизм генерирует ключи RC5 с определенной длиной в байтах, как определено атрибутом **CKA_VALUE_LEN** шаблона для данного ключа.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому ключу. Остальные атрибуты, поддерживаемые ключом типа RC5 (особенно флаги, указывающие, какие функции данный ключ поддерживает), могут быть заданы в шаблоне, или им могут быть присвоены начальные значения по умолчанию.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC5 в байтах.

12.11.5 RC5-ECB

Механизм RC5-ECB, обозначенный как **CKM_RC5_ECB**, является механизмом для одностороннего и многостороннего шифрования, упаковки ключей и распаковки ключей на основе блочного шифра RC5 компании RSA Security и режима электронной кодовой книги, как определено в FIPS PUB 81.

Данный механизм имеет параметр **CK_RC5_PARAMS**, который определяет размер слова (wordsize) и количество этапов шифрования, которые следует использовать.

Данный механизм может упаковывать и распаковывать любой секретный ключ. Конкретный носитель не всегда может упаковывать/распаковывать любой поддерживаемый им секретный ключ. Для

упаковки этот механизм зашифровывает значение атрибута **CKA_VALUE** упаковываемого ключа, добавляя на замыкающем конце до семи нулевых байт, чтобы результирующая длина была кратна размеру блока данного шифра (двойной размер слова). Данные вывода имеют ту же длину, что и дополненные данные ввода. Этот механизм не упаковывает тип ключа, длину ключа или любую иную информацию о ключе, приложение должно производить это отдельно.

Для распаковки данный механизм расшифровывает упакованный ключ, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона. Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Ограничения на типы ключей и на длину данных приведены в таблице 80.

Таблица 80 – RC5-ECB: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	RC5	Кратна размеру блока	Та же, что длина ввода	Без заключительной части
C_Decrypt	RC5	Кратна размеру блока	Та же, что длина ввода	Без заключительной части
C_WrapKey	RC5	Любая	Равна длине ввода, округленной до числа, кратного размеру блока	
C_UnwrapKey	RC5	Кратна размеру блока	Задана типом распаковываемого ключа или CKA_VALUE_LEN	

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC5 в байтах.

12.11.6 RC5-CBC

Механизм RC5-CBC, обозначенный как **CKM_RC5_CBC**, является механизмом для одностороннего и многостороннего шифрования и расшифровки, упаковки ключей и распаковки ключей на основе блочного шифра RC5 компании RSA Security и режима цепочки шифрблоков, как определно в FIPS PUB 81.

Данный механизм имеет параметр структуры **CK_RC5_CBC_PARAMS**, который задает размер слова (wordsize) и количество этапов шифрования, а также вектор инициализации для режима цепочки шифрблоков.

Данный механизм может упаковывать и распаковывать любой секретный ключ. Конкретный носитель не всегда может упаковывать/распаковывать любой поддерживаемый им секретный ключ. Для упаковки этот механизм зашифровывает значение атрибута **CKA_VALUE** упаковываемого ключа, добавляя на замыкающем конце до семи нулевых байт, чтобы результирующая длина была кратна восьми. Данные вывода имеют ту же длину, что и дополненные данные ввода. Этот механизм не упаковывает тип ключа, длину ключа или любую иную информацию о ключе, приложение должно производить это отдельно.

Для распаковки данный механизм расшифровывает упакованный ключ, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона. Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Ограничения на типы ключей и на длину данных приведены в таблице 81.

Таблицу 81 – RC5-CBC: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	RC5	Кратна размеру блока	Та же, что длина ввода	Без заключительной части
C_Decrypt	RC5	Кратна размеру блока	Та же, что длина ввода	Без заключительной части
C_WrapKey	RC5	Любая	Равна длине ввода, округленной до числа, кратного размеру блока	

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_UnwrapKey	RC5	Кратна размеру блока	Задана типом распаковываемого ключа или CKA_VALUE_LEN	

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC5 в байтах.

12.11.7 RC5-CBC с заполнением PKCS

Механизм RC5-CBC с заполнением PKCS, обозначенный как **CKM_RC5_CBC_PAD**, представляет собой механизм для одностороннего и многостороннего шифрования и расшифровки, упаковки ключей и распаковки ключей на основе блочного шифра RC5 компании RSA Security, режима цепочки шифрблоков, как определено в FIPS PUB 81 и на основе метода блочного шифра с заполнением описанного в PKCS #7.

Данный механизм имеет параметр структуры **CK_RC5_CBC_PARAMS**, который задает размер слова (wordsize) и число этапов шифрования, которые следует использовать, а также вектор инициализации для режима цепочки шифрблоков.

Имеющееся в этом механизме заполнение PKCS позволяет извлекать значение длины открытого текста из значения шифртекста. Таким образом, при распаковке ключей при помощи этого механизма, не нужно задавать значение для атрибута **CKA_VALUE_LEN**.

В дополнение к возможности упаковки и распаковки секретных ключей этот механизм может упаковывать и распаковывать секретные ключи RSA, Диффи – Хеллмана, Диффи – Хеллмана X9.42, EC (относящиеся также к ECDSA) и DSA (подробнее см. 12.6). Данные в приведенной ниже таблице по ограничениям на длину данных при упаковке и распаковке ключей не относятся к упаковке и распаковке частных секретных ключей.

Ограничения на типы ключа и длину данных приведены в таблице 82.

Таблица 82 – RC5-CBC с заполнением PKCS: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Encrypt	RC5	Любая	Равна длине ввода, округленной до числа, кратного размеру блока
C_Decrypt	RC5	Кратна размеру блока	Короче, чем длина ввода, на число между 1 и размером блока в байтах
C_WrapKey	RC5	Любая	Равна длине ввода, округленной до числа, кратного размеру блока
C_UnwrapKey	RC5	Кратна размеру блока	Короче, чем длина ввода, на число между 1 и размером блока в байтах

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC5 в байтах.

12.11.8 Механизм общей длины RC5-MAC

Механизм общей длины RC5-MAC, обозначенный как **CKM_RC5_MAC_GENERAL**, представляет собой механизм для односторонних и многосторонних операций постановки электронной подписи и верификации, основанный на блочном шифре RC5 компании RSA Security и на аутентификации данных, как определено в FIPS PUB 113.

Данный механизм имеет параметр структуры **CK_RC5_MAC_GENERAL_PARAMS**, который задает размер слова и количество этапов шифрования, которые следует применить, а также желаемую длину вывода.

Данные вывода из этого механизма берутся из начала заключительного шифрблока RC5, полученного в процессе MAC.

Ограничения на типы ключа и длину данных приведены в таблице 83.

Таблица 83 – Механизм общей длины RC2-MAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина данных
C_Sign	RC5	Любая	От 0 до размера блока, как определено в параметрах
C_Verify	RC5	Любая	От 0 до размера блока, как определено в параметрах

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC5 в байтах.

12.11.9 RC5-MAC

Механизм RC5-MAC, обозначенный как **CKM_RC5_MAC**, представляет собой частный случай механизма общей длины RC5-MAC. Вместо того, чтобы принимать параметр **CK_RC5_MAC_GENERAL_PARAMS**, этот механизм принимает параметр **CK_RC5_PARAMS**. Механизм RC5-MAC всегда производит и верифицирует MAC с размером, составляющим половину размера блока RC5.

Ограничения на типы ключа и длину данных приведены в таблице 84.

Таблица 84 – RC5-MAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	RC5	Любая	Размер слова RC5 = ⌊размер блока/2⌋
C_Verify	RC5	Любая	Размер слова RC5 = ⌊размер блока/2⌋

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа RC5 в байтах.

12.12 AES

Подробное описание стандарта AES (Advanced Encryption Standard) см. [FIPS PUB 197].

12.12.1 Определения

Данный раздел описывает ключ типа «CKK_AES» как тип CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:
CKM_AES_KEY_GEN
CKM_AES_ECB
CKM_AES_CBC
CKM_AES_MAC
CKM_AES_MAC_GENERAL
CKM_AES_CBC_PAD

12.12.2 Объекты секретного ключа AES

Объекты секретного ключа AES (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_AES**) содержат ключи AES. В таблице 85 определены атрибуты объекта секретного ключа AES в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 85 – Атрибуты секретного ключа AES

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (16, 24 или 32 байта)
CKA_VALUE_LEN [2, 3, 6]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Ниже приводится образец шаблона для создания объекта секретного ключа AES:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_AES;
CK_UTF8CHAR label[] = "Объект секретного ключа AES";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

CKA_CHECK_VALUE – это значение данного атрибута получено (выведено) из объекта ключа

посредством взятия первых трех байтов зашифрованного в режиме электронной кодовой книги одного блока нулевых (0x00) байтов с использованием задействованного по умолчанию шифра, ассоциированного с типом ключа объекта секретного ключа.

12.12.3 Генерация ключа AES

Механизм генерации ключа AES, обозначенный как **CKM_AES_KEY_GEN**, является механизмом генерации ключа стандарта Advanced Encryption Standard, утвержденного NIST.

Данный механизм не имеет параметра.

Данный механизм генерирует ключи AES с определенной длиной в байтах, как определено атрибутом **CKA_VALUE_LEN** шаблона для данного ключа.

Данный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому ключу. Остальные атрибуты, поддерживаемые ключом типа AES (особенно флаги, указывающие, какие функции данный ключ поддерживает), могут быть заданы в шаблоне данного ключа, или им могут быть присвоены начальные значения по умолчанию.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа AES в байтах.

12.12.4 AES-ECB

Механизм AES-ECB, обозначенный как **CKM_AES_ECB**, является механизмом для одностороннего и многостороннего шифрования и расшифрования, упаковки ключей и распаковки ключей на основе стандарта Advanced Encryption Standard и режима электронной кодовой книги (ECB), утвержденных NIST.

Данный механизм не имеет параметра.

Данный механизм может упаковывать и распаковывать любой секретный ключ. Конкретный носитель не всегда может упаковывать/распаковывать любой поддерживаемый им секретный ключ. Для упаковки данный механизм зашифровывает значение атрибута **CKA_VALUE** упаковываемого ключа, добавляя на замыкающем конце нулевые байты в количестве, необходимом для того, чтобы результирующая длина была кратна размеру блока данного шифра. Данные вывода имеют ту же длину, что и дополненные данные ввода. Данный механизм не упаковывает тип ключа, длину ключа или любую иную информацию о ключе, приложение должно производить это отдельно.

Для распаковки данный механизм расшифровывает упакованный ключ, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона. Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Ограничения на типы ключей и на длину данных приведены в таблице 86.

Таблица 86 – AES-ECB: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	AES	Кратна размеру блока	Та же, что и длина ввода	Без заключительной части
C_Decrypt	AES	Кратна размеру блока	Та же, что и длина ввода	Без заключительной части
C_WrapKey	AES	Любая	Равна длине ввода, округленной до числа, кратного размеру блока	
C_UnwrapKey	AES	Кратна размеру блока	Определяется типом распаковываемого ключа либо CKA_VALUE_ LEN	

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа AES в байтах.

12.12.5 AES-CBC

Механизм AES-CBC, обозначенный как **CKM_AES_CBC**, является механизмом для одностороннего и многостороннего шифрования и расшифровки, упаковки ключей и распаковки ключей на основе стандарта Advanced Encryption Standard и режима цепочки шифрблоков.

Данный механизм имеет параметр – 16-байтный вектор инициализации.

Данный механизм может упаковывать и распаковывать любой секретный ключ. Конкретный носитель не всегда может упаковывать/распаковывать любой поддерживаемый им секретный ключ. Для упаковки данный механизм зашифровывает значение атрибута **CKA_VALUE** упаковываемого ключа, добавляя на замыкающем конце нулевые байты в количестве до размера блока данного шифра ми-

нус один нулевой байт, так чтобы результирующая длина была кратна размеру блока. Данные вывода имеют ту же длину, что и дополненные данные ввода. Данный механизм не упаковывает тип ключа, длину ключа или любую иную информацию о ключе, приложение должно производить это отдельно.

Для распаковки данный механизм расшифровывает упакованный ключ, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона. Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Ограничения на типы ключей и на длину данных приведены в таблице 87.

Таблица 87 – AES-CBC: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	AES	Кратна размеру блока	Та же, что и длина ввода	Без заключительной части
C_Decrypt	AES	Кратна размеру блока	Та же, что и длина ввода	Без заключительной части
C_WrapKey	AES	Любая	Равна длине ввода, округленной до числа, кратного размеру блока	
C_UnwrapKey	AES	Кратна размеру блока	Определяется типом распаковываемого ключа либо CKA_VALUE_LEN	

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа AES в байтах.

12.12.6 AES-CBC с заполнением PKCS

Механизм AES-CBC с заполнением PKCS, обозначенный как **CKM_AES_CBC_PAD**, представляет собой механизм для одностороннего и многостороннего шифрования и расшифрования, упаковки ключей и распаковки ключей на основе стандарта Advanced Encryption Standard, утвержденного NIST, режима цепочки шифрблоков и на основе метода блочного шифра с заполнением описанного в PKCS #7.

Данный механизм имеет параметр – 16-байтный вектор инициализации.

Имеющееся в данном механизме заполнение PKCS позволяет извлекать значение длины открытого текста из значения шифртекста. Таким образом, при распаковке ключей при помощи данного механизма, не нужно задавать значение для атрибута **CKA_VALUE_LEN**.

В дополнение к возможности упаковки и распаковки секретных ключей данный механизм может упаковывать и распаковывать секретные ключи RSA, Диффи – Хеллмана, Диффи – Хеллмана X9.42, EC (относящиеся также к ECDSA) и секретные ключи DSA (подробнее см. 12.6). Данные в приведенной ниже таблице по ограничениям на длину данных при упаковке и распаковке ключей не относятся к упаковке и распаковке частных секретных ключей.

Ограничения на типы ключа и длину данных приведены в таблице 88.

Таблица 88 – AES-CBC с заполнением PKCS: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода
C_Encrypt	AES	Любая	Равна длине ввода, округленной до числа, кратного размеру блока
C_Decrypt	AES	Кратна размеру блока	Короче длины ввода на величину от 1 до длины блока в байтах
C_WrapKey	AES	Любая	Равна длине ввода, округленной до числа, кратного размеру блока
C_UnwrapKey	AES	Кратна размеру блока	Короче длины ввода на величину от 1 до длины блока в байтах

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа AES в байтах.

12.12.7 Механизм общей длины AES-MAC

Механизм общей длины AES-MAC, обозначенный как **CKM_AES_MAC_GENERAL**, представляет собой механизм для односторонних и многосторонних операций постановки электронной подписи и

верификации, основанный на стандарте Advanced Encryption Standard, утвержденном NIST, как определено в FIPS PUB 197, и на аутентификации данных, как определено в FIPS PUB 113.

Данный механизм имеет параметр структуры **CK_MAC_GENERAL_PARAMS**, который задает требуемую от механизма длину вывода.

Данные вывода из данного механизма берутся из начала заключительного шифрблока AES, полученного в процессе MAC.

Ограничения на типы ключа и длину данных приведены в таблице 89.

Таблица 89 – Механизм общей длины AES-MAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	AES	Любая	От 0 до размера блока, как определено в параметрах
C_Verify	AES	Любая	От 0 до размера блока, как определено в параметрах

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа AES в байтах.

12.12.8 AES-MAC

Механизм AES-MAC, обозначенный как **CKM_AES_MAC**, представляет собой частный случай механизма общей длины AES-MAC. Механизм AES-MAC всегда производит и верифицирует MAC с размером, составляющим половину размера блока.

Данный механизм не имеет параметра.

Ограничения на типы ключа и длину данных приведены в таблице 90.

Таблица 90 – AES-MAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	AES	Любая	½ размера блока (8 байт)
C_Verify	AES	Любая	½ размера блока (8 байт)

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа AES в байтах.

12.13 Общая информация по блочным шифрам

Для краткости механизмы для блочных шифров DES, CAST, CAST3, CAST128 (CAST5), IDEA и CDMF в настоящем стандарте будут описаны совместно. Каждый из этих шифров имеет показанные ниже механизмы, которые будут описаны в шаблонизированной (templatized) форме.

12.13.1 Определения

В настоящем разделе определяются типы ключа CKK_DES, CKK_CAST, CKK_CAST3, CKK_CAST5 (исключенный из числа рекомендованных в v2.11), CKK_CAST128, CKK_IDEA и CKK_CDMF для типа CK_KEY_TYPE, используемые в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_DES_KEY_GEN
CKM_DES_ECB
CKM_DES_CBC
CKM_DES_MAC
CKM_DES_MAC_GENERAL
CKM_DES_CBC_PAD
CKM_CDMF_KEY_GEN
CKM_CDMF_ECB
CKM_CDMF_CBC
CKM_CDMF_MAC
CKM_CDMF_MAC_GENERAL
CKM_CDMF_CBC_PAD
CKM_DES_OFB64
CKM_DES_OFB8
CKM_DES_CFB64
CKM_DES_CFB8
CKM_CAST_KEY_GEN
CKM_CAST_ECB
CKM_CAST_CBC
```

```
CKM_CAST_MAC
CKM_CAST_MAC_GENERAL
CKM_CAST_CBC_PAD
CKM_CAST3_KEY_GEN
CKM_CAST3_ECB
CKM_CAST3_CBC
CKM_CAST3_MAC
CKM_CAST3_MAC_GENERAL
CKM_CAST3_CBC_PAD
CKM_CAST5_KEY_GEN
CKM_CAST128_KEY_GEN
CKM_CAST5_ECB
CKM_CAST128_ECB
CKM_CAST5_CBC
CKM_CAST128_CBC
CKM_CAST5_MAC
CKM_CAST128_MAC
CKM_CAST5_MAC_GENERAL
CKM_CAST128_MAC_GENERAL
CKM_CAST5_CBC_PAD
CKM_CAST128_CBC_PAD
CKM_IDEA_KEY_GEN
CKM_IDEA_ECB
CKM_IDEA_CBC
CKM_IDEA_MAC
CKM_IDEA_MAC_GENERAL
CKM_IDEA_CBC_PAD
```

12.13.2 Объекты секретного ключа DES

Объекты секретного ключа DES (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_DES**) содержат ключи DES единой длины (single-length). В таблице 91 определены атрибуты объекта секретного ключа DES в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 91 – Объект секретного ключа DES

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (всегда длиной 8 байт)
См. сноски в таблице 15.		

Ключи DES всегда должны иметь верно расположенными свои биты четности, как описано в FIPS PUB 46-3. Попытка создать или распаковать ключ DES с неверной четностью приведет к выдаче сообщения об ошибке.

Ниже приводится образец шаблона для создания объекта секретного ключа DES:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_DES;
CK_UTF8CHAR label[] = “Объект секретного ключа DES”;
CK_BYTE value[8] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

CKA_CHECK_VALUE – значение данного атрибута получено из объекта ключа посредством взятия первых трех байт из результата шифрования в режиме электронной кодовой книги одного

блока нулевых (0x00) байт с использованием применяемого по умолчанию шифра, ассоциированного с типом ключа объекта секретного ключа.

12.13.3 Объекты секретного ключа CAST

Объекты секретного ключа CAST (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_CAST**) содержат ключи CAST. В таблице 92 определены атрибуты объекта секретного ключа CAST в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 92 – Атрибуты объекта секретного ключа CAST

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (от 1 до 8 байт)
CKA_VALUE_LEN [2, 3, 6]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Ниже приводится образец шаблона для создания объекта секретного ключа CAST:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAST;
CK_UTF8CHAR label[] = "Объект секретного ключа CAST";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.13.4 Объекты секретного ключа CAST3

Объекты секретного ключа CAST3 (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_CAST**) содержат ключи CAST3. В таблице 93 определены атрибуты объекта секретного ключа CAST3 в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 93 – Атрибуты объекта секретного ключа CAST3

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (от 1 до 8 байт)
CKA_VALUE_LEN [2, 3, 6]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Ниже приводится образец шаблона для создания объекта секретного ключа CAST3:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAST3;
CK_UTF8CHAR label[] = "Объект секретного ключа CAST3";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.3.5 Объекты секретного ключа CAST128 (CAST5)

Объекты секретного ключа CAST128 (известного также, как CAST5) (класс объекта –

CKO_SECRET_KEY, тип ключа – CKK_CAST128 или **CKK_CAST5**) содержат ключи CAST128. В таблице 94 определены атрибуты объекта секретного ключа CAST128 в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 94 – Атрибуты объекта секретного ключа CAST128 (CAST5)

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (от 1 до 16 байт)
CKA_VALUE_LEN [2, 3, 6]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Ниже приводится образец шаблона для создания объекта секретного ключа CAST128 (CAST5):

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAST128;
CK_UTF8CHAR label[] = “Объект секретного ключа CAST128”;
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.13.6 Объекты секретного ключа IDEA

Объекты секретного ключа IDEA (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_IDEA**) содержат ключи IDEA. В таблице 95 определены атрибуты объекта секретного ключа IDEA в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 95 – Объект секретного ключа IDEA

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (всегда длиной 16 байт)
См. сноски в таблице 15.		

Ниже приводится образец шаблона для создания объекта секретного ключа IDEA:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_IDEA;
CK_UTF8CHAR label[] = “Объект секретного ключа IDEA”;
CK_BYTE value[16] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.13.7 Объекты секретного ключа CDMF

Объекты секретного ключа CDMF (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_CDMF**) содержат ключи CDMF единой длины (single-length). В таблице 96 определены атрибуты объекта секретного ключа CDMF в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 96 – Объект секретного ключа CDMF

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (всегда длиной 8 байт)
См. сноски в таблице 15.		

Ключи CDMF всегда должны иметь верно расположенными свои биты четности, которые должны быть представлены в том же виде, как указано для ключей DES в FIPS PUB 46-3. Попытка создать или распаковать ключ CDMF с неверной четностью приведет к выдаче сообщения об ошибке.

Ниже приводится образец шаблона для создания объекта секретного ключа CDMF:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CDMF;
CK_UTF8CHAR label[] = "Объект секретного ключа CDMF";
CK_BYTE value[8] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.13.8 Общие параметры механизма блочного шифра

♦ **CK_MAC_GENERAL_PARAMS; CK_MAC_GENERAL_PARAMS_PTR**

CK_MAC_GENERAL_PARAMS является структурой, которая представляет параметры для механизмов общей длины MAC шифров DES, DES3 (тройной DES, трайпл-DES), CAST, CAST3, CAST128 (CAST5), IDEA, CDMF и AES. Он также представляет параметры для механизмов общей длины HMAC (т. е. MD2, MD5, SHA-1, SHA-256, SHA-384, SHA-512, RIPEMD-128 и RIPEMD-160), а также для двух механизмов SSL 3.0 MAC (т. е. MD5 и SHA-1). Данная структура содержит длину результата применения MAC, производимого этими механизмами. Структура задается следующим образом:

```
typedef CK_ULONG CK_MAC_GENERAL_PARAMS;
```

CK_MAC_GENERAL_PARAMS_PTR является указателем на **CK_MAC_GENERAL_PARAMS**.

12.13.9 Общая информация по генерации ключа блочного шифра

Шифр <НАИМЕНОВАНИЕ> имеет механизм генерации ключа «<НАИМЕНОВАНИЕ> key generation», обозначенный как **CKM_<НАИМЕНОВАНИЕ>_KEY_GEN.**

Данный механизм не имеет параметра.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому ключу. Остальные атрибуты, поддерживаемые ключом данного типа (особенно флаги, указывающие, какие функции данный ключ поддерживает), могут быть заданы в шаблоне данного ключа, или им
могут быть присвоены начальные значения по умолчанию.

Когда генерируются ключи DES или CDMF, их биты четности располагаются верно, как указано в FIPS PUB 46-3. Аналогичным образом, когда генерируется ключ тройного DES, к каждому из составляющих его ключей DES биты четности располагаются верно.

Когда генерируются ключи DES или CDMF, именно от носителя зависит, возможна ли генерация «слабых» или «полуслабых» ключей. Аналогичным образом, когда генерируются ключи тройного DES, именно от носителя зависит, возможно ли, чтобы какие-либо из компонентов ключей DES представляли собой «слабые» или «полуслабые» ключи.

Когда генерируются ключи CAST, CAST3 или CAST128 (CAST5), шаблон секретного ключа должен задать атрибут **CKA_VALUE_LEN**.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* могут использоваться или могут не использоваться. Шифры CAST, CAST3 и CAST128 (CAST5) имеют изменяемые размеры ключа, поэтому поля *ulMinKeySize* и *ulMaxKeySize* стркутуры **CK_MECHANISM_INFO** задают поддерживаемый диапазон размеров ключей в байтах для механизмов генерации ключей данных шифров. Для шифров DES, DES3 (тройной DES), IDEA и CDMF данные поля не используются.

12.13.10 Общая информация по блочному шифру с ECB

Шифр <НАИМЕНОВАНИЕ> имеет механизм электронной кодовой книги «<НАИМЕНОВАНИЕ>-ECB», обозначенный как **CKM_<НАИМЕНОВАНИЕ>_ECB**. Это механизм для одностороннего и многостороннего шифрования и расшифрования, упаковки и распаковки ключа <НАИМЕНОВАНИЕ>.

Данный механизм не имеет параметра.

Данный механизм может упаковывать и распаковывать любой секретный ключ. Конкретный носитель не всегда может упаковывать/распаковывать любой поддерживаемый им секретный ключ. Для упаковки этот механизм зашифровывает значение атрибута **CKA_VALUE** упаковываемого ключа, добавляя на замыкающем конце нулевые байты в количестве до размера блока данного шифра минус один нулевой байт так, чтобы результирующая длина была кратна размеру блока шифра <НАИМЕНОВАНИЕ>. Данные вывода имеют ту же длину, что и дополненные данные ввода. Механизм не упаковывает тип ключа, длину ключа или любую иную информацию о ключе, приложение должно производить это отдельно.

Для распаковки данный механизм расшифровывает упакованный ключ, затем усекает результат в соответствии с атрибутом **CKA_KEY_TYPE** шаблона и (если он у него есть и если данный тип ключа его поддерживает) в соответствии с атрибутом **CKA_VALUE_LEN** шаблона. Данный механизм выдает результат как атрибут **CKA_VALUE** нового ключа; другие атрибуты, требуемые для данного типа ключа, должны быть заданы в шаблоне.

Ограничения на типы ключа и длину данных приведены в таблице 97.

Таблица 97 – Общая информация по блочному шифру с ECB: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	<НАИМЕНОВАНИЕ>	Кратна размеру блока	Та же, что и длина ввода	Без заключительной части
C_Decrypt	<НАИМЕНОВАНИЕ>	Кратна размеру блока	Та же, что и длина ввода	Без заключительной части
C_WrapKey	<НАИМЕНОВАНИЕ>	Любая	Равна длине ввода, округленной до числа, кратного размеру блока	
C_UnwrapKey	<НАИМЕНОВАНИЕ>	Любая	Определяется типом распаковываемого ключа либо CKA_VALUE_LEN	

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* могут использоваться или могут не использоваться. Шифры CAST, CAST3 и CAST128 (CAST5) имеют изменяемые размеры ключа, поэтому поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** задают поддерживаемый диапазон размеров ключей в байтах для механизмов генерации ключей данных шифров. Для шифров DES, DES3 (тройной DES), IDEA и CDMF данные поля не используются.

12.13.11 Общая информация по блочному шифру с CBC

Шифр <НАИМЕНОВАНИЕ> имеет режим цепочки шифрблоков «<НАИМЕНОВАНИЕ>-CBC», обозначенный как **CKM_<НАИМЕНОВАНИЕ>_CBC**. Это механизм для одностороннего и многостороннего шифрования и расшифрования, упаковки и распаковки ключа <НАИМЕНОВАНИЕ>.

Данный шифр имеет параметр, вектор инициализации для режима цепочки шифрблоков. Вектор инициализации имеет ту же длину, что и размер блоков шифра <НАИМЕНОВАНИЕ>.

Ограничения на типы ключей и длину данных приведены в таблице 98.

Таблица 98 – Общая информация по блочному шифру с CBC: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	<НАИМЕНОВАНИЕ>	Кратна размеру блока	Та же, что и длина ввода	Без заключительной части
C_Decrypt	<НАИМЕНОВАНИЕ>	Кратна размеру блока	Та же, что и длина ввода	Без заключительной части
C_WrapKey	<НАИМЕНОВАНИЕ>	Любая	Равна длине ввода, округленной до числа, кратного размеру блока	
C_UnwrapKey	<НАИМЕНОВАНИЕ>	Любая	Определяется типом распаковываемого ключа либо CKA_VALUE_LEN	

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* могут не использоваться. Шифры CAST, CAST3 и CAST128 (CAST5) имеют изменяемые размеры ключа, поэтому для этих шифров поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** задают поддерживаемый диапазон размеров ключей в байтах для механизмов генерации ключей. Для шифров DES, DES3 (тройной DES), IDEA и CDMF данные поля не используются.

12.13.12 Общая информация по блочному шифру с CBC и заполнением PKCS

Шифр <НАИМЕНОВАНИЕ> имеет режим цепочки шифрблоков с заполнением PKCS «<НАИМЕНОВАНИЕ>-CBC с заполнением PKCS», обозначенный как **CKM_<NAME>_CBC_PAD**. Это механизм одностороннего и двустороннего шифрования и расшифрования, упаковки и распаковки ключей посредством <НАИМЕНОВАНИЕ>. Весь шифртекст заполняется посредством заполнения PKCS.

Данный шифр имеет параметр – вектор инициализации для режима цепочки шифрблоков. Данный вектор инициализации имеет ту же длину, что и размер блока шифра <НАИМЕНОВАНИЕ>.

Заполнение PKCS в данном механизме позволяет восстанавливать значение длины открытого текста из значения шифртекста. Таким образом, при распаковке ключей посредством данного механизма не следует задавать значение для атрибута **CKA_VALUE_LEN**.

В дополнение к способности запаковывать и распаковывать секретные ключи данный механизм может запаковывать и распаковывать секретные ключи RSA, Диффи – Хеллмана, Диффи – Хеллмана X9.42, EC (имеющие также отношение к ECDSA) и DSA (подробнее см. 12.6). Данные в приведенной ниже таблице по ограничениям длины при упаковке и распаковке ключей не относятся к упаковке и распаковке секретных ключей.

Ограничения на типы ключей и длину данных приведены в таблице 99.

Таблица 99 – Общая информация по блочному шифру с CBC и заполнением PKCS: Длина ключа и данных

Функция	Тип данных	Длина ввода	Длина вывода
C_Encrypt	<НАИМЕНОВАНИЕ>	Любая	Равна длине ввода, округленной до числа, кратного размеру блока
C_Decrypt	<НАИМЕНОВАНИЕ>	Кратна размеру блока	Короче длины ввода на величину от 1 до длины блока в байтах
C_WrapKey	<НАИМЕНОВАНИЕ>	Любая	Равна длине ввода, округленной до числа, кратного размеру блока
C_UnwrapKey	<НАИМЕНОВАНИЕ>	Кратна размеру блока	Короче длины ввода на величину от 1 до длины блока в байтах

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* могут использоваться или могут не использоваться. Шифры CAST, CAST3 и CAST128 (CAST5) имеют изменяемые размеры ключа, поэтому поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** задают поддерживаемый диапазон размеров ключей в байтах для механизмов генерации ключей данных шифров. Для шифров DES, DES3 (тройной DES), IDEA и CDMF данные поля не используются.

12.13.13 Общая информация по шифру общей длины с MAC

Шифр <НАИМЕНОВАНИЕ> имеет режим общей длины MAC – «General-length <НАИМЕНОВАНИЕ>-MAC», обозначенный **CKM_<NAME>_MAC_GENERAL**. Это механизм для односторонних и многосторонних операций постановки электронной подписи и верификации, основанных на алгоритме шифрования <НАИМЕНОВАНИЕ> и аутентификации данных, как это определено в FIPS PUB 113.

Данный шифр имеет параметр **CK_MAC_GENERAL_PARAMS**, который задает размер вывода.

Данные вывода из этого механизма берутся из начала заключительного шифрблока, полученного в процессе MAC.

Ограничения на типы ключа и длину данных приведены в таблице 100.

Таблица 100 – Общая информация по шифру общей длины с MAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	<НАИМЕНОВАНИЕ>	Любая	От 0 до размера блока, в зависимости от параметров
C_Verify	<НАИМЕНОВАНИЕ>	Любая	От 0 до размера блока, в зависимости от параметров

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* могут использоваться или могут не использоваться. Шифры CAST, CAST3 и CAST128 (CAST5) имеют изменяемые размеры ключа, поэтому поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** задают поддерживаемый диапазон размеров ключей в байтах для механизмов генерации ключей данных шифров. Для шифров DES, DES3 (тройной DES), IDEA и CDMF данные поля не используются.

12.13.14 Общая информация по блочному шифру с MAC

Шифр <НАИМЕНОВАНИЕ> имеет механизм MAC – «<НАИМЕНОВАНИЕ>-MAC», обозначенный как **CKM_<NAME>_MAC**. Данный механизм представляет собой частный случай описанного выше механизма **CKM_<NAME>_MAC_GENERAL**. Он всегда производит вывод с размером наполовину меньше размера блока шифра <НАИМЕНОВАНИЕ>.

Данный механизм не имеет параметров.

Ограничения на типы ключа и длину данных приведены в таблице 101.

Таблица 101 – Общая информация по блочному шифру с MAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	<НАИМЕНОВАНИЕ>	Любая	⌊размер блока/2⌋
C_Verify	<НАИМЕНОВАНИЕ>	Любая	⌊размер блока/2⌋

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* могут использоваться или могут не использоваться. Шифры CAST, CAST3 и CAST128 (CAST5) имеют изменяемые размеры ключа, поэтому поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** задают поддерживаемый диапазон размеров ключей в байтах для механизмов генерации ключей данных шифров. Для шифров DES, DES3 (тройной DES), IDEA и CDMF данные поля не используются.

12.14 Деривация ключей посредством шифрования данных – DES и AES

Данные механизмы позволяют осуществлять деривацию ключей с использованием такого результата операции шифрования, как величина ключа. Данные механизмы предназначены для использо-вания с функцией C_DeriveKey function.

12.14.1 Определения

Механизмы:

```
CKM_DES_ECB_ENCRYPT_DATA
CKM_DES_CBC_ENCRYPT_DATA
CKM_DES3_ECB_ENCRYPT_DATA
CKM_DES3_CBC_ENCRYPT_DATA
CKM_AES_ECB_ENCRYPT_DATA
CKM_AES_CBC_ENCRYPT_DATA
typedef struct CK_DES_CBC_ENCRYPT_DATA_PARAMS {
  CK_BYTE iv[8];
  CK_BYTE_PTR pData;
  CK_ULONG length;
```

```
} CK_DES_CBC_ENCRYPT_DATA_PARAMS;
typedef CK_DES_CBC_ENCRYPT_DATA_PARAMS
        CK_PTR CK_DES_CBC_ENCRYPT_DATA_PARAMS_PTR;

typedef struct CK_AES_CBC_ENCRYPT_DATA_PARAMS {
  CK_BYTE iv[16];
  CK_BYTE_PTR pData;
  CK_ULONG length;
} CK_AES_CBC_ENCRYPT_DATA_PARAMS;
typedef CK_DES_CBC_ENCRYPT_DATA_PARAMS CK_PTR
        CK_DES_CBC_ENCRYPT_DATA_PARAMS_PTR;
```

12.14.2 Параметры механизма

Использует CK_KEY_DERIVATION_STRING_DATA, как определено в 12.34.2.

Таблица 102 – Параметры механизма

CKM_DES_ECB_ENCRYPT_DATA CKM_DES3_ECB_ENCRYPT_DATA	Использует структуру CK_KEY_DERIVATION_STRING_DATA. Параметром являются данные, подлежащие шифрованию, его длина в байтах должна быть выражена числом, кратным 8
CKM_AES_ECB_ENCRYPT_DATA	Использует структуру CK_KEY_DERIVATION_STRING_DATA. Параметром являются данные, подлежащие шифрованию, его длина должна быть выражена числом, кратным 16
CKM_DES_CBC_ENCRYPT_DATA CKM_DES3_CBC_ENCRYPT_DATA	Использует CK_DES_CBC_ENCRYPT_DATA_PARAMS. Параметром является 8-байтный вектор инициализации (IV) с последующими данными. Значение данных должно быть кратно 8
CKM_AES_CBC_ENCRYPT_DATA	Использует CK_AES_CBC_ENCRYPT_DATA_PARAMS. Параметром является 16-байтный вектор инициализации (IV) с последующими данными. Значение данных должно быть кратно 16

12.14.3 Описание механизма

Данные механизмы будут функционировать посредством осуществления шифрования на данных, предоставленных с использованием базового ключа. Результирующий шифртекст будет использован для создания ключевого значения результирующего ключа. Если будет использоваться не весь шифр-текст, тогда неиспользуемой частью станет хвостовой участок (самые младшие по разрядности байты) данных шифртекста. Получаемый таким образом ключ будет задан шаблоном атрибута, представленным и ограниченным длиной шифртекста, доступного для значения ключа, а также обычными ограничениями на деривацию, описанными в PKCS11.

Управление шаблоном атрибута, задание атрибута по умолчанию и подготовка значения ключа будут осуществляться в соответствии с механизмом деривации ключа SHA-1, описанным в 12.21.5.

Если данные будут слишком короткими для создания требуемого ключа, тогда механизм выдаст сообщение CKR_DATA_LENGTH_INVALID.

12.15 DES двойной и тройной длины

12.15.1 Определения

В данной секции определяется тип ключей CKK_DES2 и CKK_DES3 как тип CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_DES2_KEY_GEN
CKM_DES3_KEY_GEN
CKM_DES3_ECB
CKM_DES3_CBC
CKM_DES3_MAC
CKM_DES3_MAC_GENERAL
CKM_DES3_CBC_PAD
```

12.15.2 Объекты секретного ключа DES2

Объекты секретного ключа DES2 (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_DES2**) содержат ключи DES двойной длины. В таблице 103 определены атрибуты объекта секретного ключа DES2 в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 103 – Атрибуты объекта секретного ключа DES2

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (всегда длиной 16 байт)
См. сноски в таблице 15.		

Ключи DES2 всегда должны иметь верно расположенными свои биты четности, как описано в FIPS PUB 46-3 (т. е. каждый из ключей DES, составляющих ключ DES2, должен иметь верно расположенными свои биты четности). Попытка создать или распаковать ключ DES2 с неверной четностью приведет к выдаче сообщения об ошибке.

Ниже приводится образец шаблона для создания объекта секретного ключа DES двойной длины:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_DES2;
CK_UTF8CHAR label[] = “Объект секретного ключа DES2”;
CK_BYTE value[16] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

CKA_CHECK_VALUE – значение данного атрибута получено из объекта ключа посредством взятия первых трех байт из результата шифрования в режиме электронной кодовой книги одного блока нулевых (0x00) байт с использованием применяемого по умолчанию шифра, ассоциированного с типом ключа объекта секретного ключа.

12.15.3 Объекты секретного ключа

Объекты секретного ключа DES3 (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_DES3**) содержат ключи DES тройной длины. В таблице 104 определены атрибуты объекта секретного ключа DES3 в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 104 – Атрибуты объекта секретного ключа DES3

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байт	Значение ключа (всегда длиной 24 байта)
См. сноски в таблице 15.		

Ключи DES3 всегда должны иметь верно расположенными свои биты четности, как описано в FIPS PUB 46-3 (т. е. каждый из ключей DES, составляющих ключ DES3, должен иметь верно расположенными свои биты четности). Попытка создать или распаковать ключ DES3 с неверной четностью приведет к выдаче сообщения об ошибке.

Ниже приводится образец шаблона для создания объекта секретного ключа DES тройной длины:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_DES3;
CK_UTF8CHAR label[] = “Объект секретного ключа DES3”;
CK_BYTE value[24] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
```

```
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

CKA_CHECK_VALUE – значение данного атрибута получено из объекта ключа посредством взятия первых трех байт из результата шифрования в режиме электронной кодовой книги одного блока нулевых (0x00) байт с использованием применяемого по умолчанию шифра, ассоциированного с типом ключа объекта секретного ключа.

12.15.4 Генерация ключа DES двойной длины

Механизм генерации ключа DES двойной длины, обозначенный как **CKM_DES2_KEY_GEN**, является механизмом генерации ключа для ключей DES двойной длины. Ключи DES, составляющие ключи DES двойной длины, оба имеют верно расположенными свои биты четности, как указано в FIPS PUB 46-3.

Данный механизм не имеет параметра.

Указанный механизм добавляет атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** к новому ключу. Остальные атрибуты, поддерживаемые ключом типа DES двойной длины (особенно флаги, указывающие, какие функции данный ключ поддерживает), могут быть заданы в шаблоне данного ключа, или им могут быть присвоены начальные значения по умолчанию.

Ключи DES двойной длины могут быть использованы с теми же механизмами, что и ключи DES тройной длины: **CKM_DES3_ECB**, **CKM_DES3_CBC**, **CKM_DES3_CBC_PAD**, **CKM_DES3_MAC_GENERAL** и **CKM_DES3_MAC** (эти механизмы описаны в шаблонной форме в разделе 12.13). Шифрование тройным DES с ключом DES двойной длины эквивалентно шифрованию ключом DES тройной длины с K1 = K3, как описано в FIPS PUB 46-3.

Когда генерируются ключи DES двойной длины, именно от носителя зависит, могут ли какие-либо из составляющих их ключей DES быть «слабыми» или «полуслабыми» ключами.

12.15.5 Порядок операций DES тройной длины

Процедуры шифрования с применением DES тройной длины проводятся, как это описано в FIPS PUB 46-3: зашифровывание, расшифровка, зашифровывание. Процедуры расшифровки осуществляются посредством трех шагов в обратном направлении: расшифровки, зашифровывания, расшифровки. Математическое представление операций зашифровывания и расшифровки выглядят следующим образом:

DES3-E({K1,K2,K3}, P) = E(K3, D(K2, E(K1, P)))

DES3-D({K1,K2,K3}, C) = D(K1, E(K2, D(K3, P)))

12.15.6 DES тройной длины в режиме CBC

Операции DES тройной длины в режиме CBC с ключами двойной или тройной длины осуществляются с использованием внешнего CBC, как определено в X9.52. X9.52 описывает этот режим как TCBC. Математическое представление операции зашифровывания и расшифровки CBC выглядит следующим образом:

DES3-CBC-E({K1,K2,K3}, P) = E(K3, D(K2, E(K1, P + I)))

DES3-CBC-D({K1,K2,K3}, C) = D(K1, E(K2, D(K3, P))) + I

Значение *I* является либо 8-байтным вектором инициализации, либо предыдущим блоком шифртекста, который прибавляется к текущему блоку ввода. Применяемая операция сложения – сложение по модулю 2 (XOR).

12.15.7 DES и DES тройной длины в режиме OFB

Шифр DES имеет режим выходной обратной связи (Output-feedback, OFB), DES-OFB, обозначаемый как **CKM_DES_OFB8** и **CKM_DES_OFB64**. Это механизм для одностороннего и многостороннего шифрования и расшифровки с использованием DES.

Данный механизм имеет параметр – вектор инициализации для данного режима. Данный вектор инициализации имеет длину, совпадающую с размером блока.

Ограничения на типы ключа и на длину данных приведены в таблице 105.

Таблица 105 – OFB: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	CKK_DES, CKK_DES2, CKK_DES3	Любая	Та же, что и длина ввода	Без заключительной части
C_Decrypt	CKK_DES, CKK_DES2, CKK_DES3	Любая	Та же, что и длина ввода	Без заключительной части

Для данного механизма структура **CK_MECHANISM_INFO** такая же, как это определено для режима CBC.

12.15.8 DES и тройной DES в режиме CFB

Шифр DES имеет режим обратной связи по шифру (cipher-feedback, CFB), DES-CFB, обозначаемый как **CKM_DES_CFB8** и **CKM_DES_CFB64**. Это механизм для одностороннего и многостороннего шифро-вания и расшифровки с использованием DES.

Данный механизм имеет параметр – вектор инициализации для данного режима. Данный вектор инициализации имеет длину, совпадающую с размером блока.

Ограничения на типы ключа и на длину данных приведены в таблице 106.

Таблица 106 – CFB: Длина ключа и данных

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	CKK_DES, CKK_DES2, CKK_DES3	Любая	Та же, что и длина ввода	Без заключительной части
C_Decrypt	CKK_DES, CKK_DES2, CKK_DES3	Любая	Та же, что и длина ввода	Без заключительной части

Для данного механизма структура **CK_MECHANISM_INFO** такая же, как это определено для режима.

12.16 SKIPJACK

12.16.1 Определения

В настоящем разделе определяется тип ключа CKK_SKIPJACK как CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_SKIPJACK_KEY_GEN
CKM_SKIPJACK_ECB64
CKM_SKIPJACK_CBC64
CKM_SKIPJACK_OFB64
CKM_SKIPJACK_CFB64
CKM_SKIPJACK_CFB32
CKM_SKIPJACK_CFB16
CKM_SKIPJACK_CFB8
CKM_SKIPJACK_WRAP
CKM_SKIPJACK_PRIVATE_WRAP
CKM_SKIPJACK_RELAYX
```

12.16.2 Объекты секретного ключа SKIPJACK

Объекты секретного ключа SKIPJACK (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_SKIPJACK**) содержат ключ единой длины MEK или TEK. В таблице 107 определены атрибуты объекта секретного ключа SKIPJACK в дополнение к общим атрибутам, определенным для данного класса объектов.

Таблица 107 – Объект секретного ключа SKIPJACK

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (всегда длиной 12 байт)
См. сноски в таблице 15.		

Ключи SKIPJACK имеют 16 бит контрольной суммы, и эти биты должны быть верно размещены. Попытка создать или распаковать ключ SKIPJACK с неверной контрольной суммой приведут к выдаче сообщения об ошибке.

Неясно, существует ли (и будет ли вообще существовать) носитель, который позволит своему программному приложению создать ключ SKIPJACK с заданным значением. Тем не менее приводим шаблоны, предназначенные для этой цели.

Ниже приведен шаблон для создания объекта секретного ключа SKIPJACK MEK:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SKIPJACK;
CK_UTF8CHAR label[] = “Объект секретного ключа SKIPJACK MEK”;
CK_BYTE value[12] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},

  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

Далее приведен шаблон для создания объекта секретного ключа SKIPJACK TEK:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SKIPJACK;
CK_UTF8CHAR label[] = “A SKIPJACK TEK secret key object”;
CK_BYTE value[12] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_WRAP, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.16.3 Параметры механизма SKIPJACK

♦ **CK_SKIPJACK_PRIVATE_WRAP_PARAMS; CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR**

CK_SKIPJACK_PRIVATE_WRAP_PARAMS является структурой, представляющей параметры для механизма **CKM_SKIPJACK_PRIVATE_WRAP**. Она задается следующим образом:

```
typedef struct CK_SKIPJACK_PRIVATE_WRAP_PARAMS {
  CK_ULONG ulPasswordLen;
  CK_BYTE_PTR pPassword;
  CK_ULONG ulPublicDataLen;
  CK_BYTE_PTR pPublicData;
  CK_ULONG ulPandGLen;
  CK_ULONG ulQLen;
  CK_ULONG ulRandomLen;
  CK_BYTE_PTR pRandomA;
  CK_BYTE_PTR pPrimeP;
  CK_BYTE_PTR pBaseG;
```

```
  CK_BYTE_PTR pSubprimeQ;
} CK_SKIPJACK_PRIVATE_WRAP_PARAMS;
```

Поля данной структуры имеют следующие значения:

ulPasswordLen длина пароля;

pPassword указатель на буфер, содержащий пароль, предоставленный пользователем;

ulPublicDataLen размер открытого ключа другой стороны в системе обмена ключами;

pPublicData указатель на значение открытого ключа другой стороны в системе обмена ключами;

ulPandGLen длина первичного и базового значений;

ulQLen длина предпервичного значения;

ulRandomLen размер в байтах случайной величины Ra;

pRandomA указатель на данные Ra;

pPrimeP указатель на значение базового p;

pBaseG указатель на значение базового g;

pSubprimeQ указатель на значение предпервичного q.

CK_SKIPJACK_PRIVATE_WRAP_PARAMS_PTR является указателем на **CK_PRIVATE_WRAP_PARAMS**.

♦ **CK_SKIPJACK_RELAYX_PARAMS; CK_SKIPJACK_RELAYX_PARAMS_PTR**

CK_SKIPJACK_RELAYX_PARAMS является структурой, представляющей параметры для механизма **CKM_SKIPJACK_RELAYX**. Она задается следующим образом:

```
typedef struct CK_SKIPJACK_RELAYX_PARAMS {
  CK_ULONG ulOldWrappedXLen;
  CK_BYTE_PTR pOldWrappedX;
  CK_ULONG ulOldPasswordLen;
  CK_BYTE_PTR pOldPassword;
  CK_ULONG ulOldPublicDataLen;
  CK_BYTE_PTR pOldPublicData;
  CK_ULONG ulOldRandomLen;
  CK_BYTE_PTR pOldRandomA;
  CK_ULONG ulNewPasswordLen;
  CK_BYTE_PTR pNewPassword;
  CK_ULONG ulNewPublicDataLen;
  CK_BYTE_PTR pNewPublicData;
  CK_ULONG ulNewRandomLen;
  CK_BYTE_PTR pNewRandomA;
} CK_SKIPJACK_RELAYX_PARAMS;
```

Поля данной структуры имеют следующие значения:

ulOldWrappedXLen длина в байтах старого упакованного ключа;

pOldWrappedX указатель на старый упакованный ключ;

ulOldPasswordLen длина старого пароля;

pOldPassword указатель на буфер, содержащий старый пароль, предоставленный пользователем;

ulOldPublicDataLen размер открытого ключа для обмена ключами;

pOldPublicData указатель на значение старого открытого ключа для обмена ключами;

ulOldRandomLen размер старой случайной величины Ra в байтах;

pOldRandomA указатель на старые данные Ra;

ulNewPasswordLen длина нового пароля;

pNewPassword указатель на буфер, содержащий новый пароль, предоставленный пользователем;

ulNewPublicDataLen размер нового открытого ключа для обмена ключами;

pNewPublicData указатель на значение ногово открытого ключа для обмена ключами;

ulNewRandomLen размер новой случайной величины Ra в байтах;

pNewRandomA указатель на данные нового Ra.

CK_SKIPJACK_RELAYX_PARAMS_PTR является указателем на **CK_SKIPJACK_RELAYX_PARAMS**.

12.16.4 Генерация ключа SKIPJACK

Механизм генерации ключа SKIPJACK, именуемый **CKM_SKIPJACK_KEY_GEN**, является механизмом генерации ключа для SKIPJACK. На выходе данного механизма получается величина, именуемая ключ шифрования сообщения (Message Encryption Key, MEK).

Данный механизм не имеет параметра.

Данный механизм добавляет к новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE**.

12.16.5 SKIPJACK-ECB64

Механизм SKIPJACK-ECB64, обозначенный как **CKM_SKIPJACK_ECB64**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением SKIPJACK в режиме 64-битной электронной кодовой книги, как определено в FIPS PUB 185.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 108.

Таблица 108 – SKIPJACK-ECB64: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	SKIPJACK	Кратна 8	Та же, что и длина ввода	Без заключительной части
C_Decrypt	SKIPJACK	Кратна 8	Та же, что и длина ввода	Без заключительной части

12.16.6 SKIPJACK-CBC64

SKIPJACK-CBC64, обозначенный как **CKM_SKIPJACK_CBC64**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением SKIPJACK в режиме 64-битной цепочки шифрблоков, как определено в FIPS PUB 185.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 109.

Таблица 109 – SKIPJACK-CBC64: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	SKIPJACK	Кратна 8	Та же, что и длина ввода	Без заключительной части
C_Decrypt	SKIPJACK	Кратна 8	Та же, что и длина ввода	Без заключительной части

12.16.7 SKIPJACK-OFB64

SKIPJACK-OFB64, обозначенный как **CKM_SKIPJACK_OFB64**, является механизмом для одностороннего и многостороннего шифрования и расшифрования при помощи SKIPJACK в 64-битном режиме выходной обратной связи (Output-feedback, OFB), как определено в FIPS PUB 185.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 110.

Таблица 110 – SKIPJACK-OFB64: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	SKIPJACK	Кратна 8	Та же, что и длина ввода	Без заключительной части
C_Decrypt	SKIPJACK	Кратна 8	Та же, что и длина ввода	Без заключительной части

12.16.8 SKIPJACK-CFB64

SKIPJACK-CFB64, обозначенный как **CKM_SKIPJACK_CFB64**, представляет собой механизм для одностороннего и многостороннего шифрования и расшифрования при помощи SKIPJACK 64-битном режиме обратной связи по шифру (cipher-feedback, CFB), как определено в FIPS PUB 185.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 111.

Таблица 111 – SKIPJACK-CFB64: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	SKIPJACK	Кратна 8	Та же, что и длина ввода	Без заключительной части
C_Decrypt	SKIPJACK	Кратна 8	Та же, что и длина ввода	Без заключительной части

12.16.9 SKIPJACK-CFB32

SKIPJACK-CFB32, обозначенный как **CKM_SKIPJACK_CFB32**, представляет собой механизм для одностороннего и многостороннего шифрования и расшифровки при помощи SKIPJACK 32-битном режиме обратной связи по шифру (cipher-feedback, CFB), как определено в FIPS PUB 185.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 112.

Таблица 112 – SKIPJACK-CFB32: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	SKIPJACK	Кратна 4	Та же, что и длина ввода	Без заключительной части
C_Decrypt	SKIPJACK	Кратна 4	Та же, что и длина ввода	Без заключительной части

12.16.10 SKIPJACK-CFB16

SKIPJACK-CFB16, обозначенный как **CKM_SKIPJACK_CFB16**, представляет собой механизм для одностороннего и многостороннего шифрования и расшифровки при помощи SKIPJACK 16-битном режиме обратной связи по шифру (cipher-feedback, CFB), как определено в FIPS PUB 185.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 113.

Таблица 113 – SKIPJACK-CFB16: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	SKIPJACK	Кратна 4	Та же, что и длина ввода	Без заключительной части
C_Decrypt	SKIPJACK	Кратна 4	Та же, что и длина ввода	Без заключительной части

12.16.11 SKIPJACK-CFB8

SKIPJACK-CFB8, обозначенный как **CKM_SKIPJACK_CFB8**, представляет собой механизм для одностороннего и многостороннего шифрования и расшифровки при помощи SKIPJACK 8-битном режиме обратной связи по шифру (cipher-feedback, CFB), как определено в FIPS PUB 185.

Данный механизм имеет параметр 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 114.

Таблица 114 – SKIPJACK-CFB8: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	SKIPJACK	Кратна 4	Та же, что и длина ввода	Без заключительной части
C_Decrypt	SKIPJACK	Кратна 4	Та же, что и длина ввода	Без заключительной части

12.16.12 SKIPJACK-WRAP

Механизм SKIPJACK-WRAP, обозначенный как **CKM_SKIPJACK_WRAP**, используется для упаковки и распаковки секретного ключа MEK. Он может упаковывать и распаковывать ключи SKIPJACK, BATON и JUNIPER.

Данный механизм не имеет параметра.

12.16.13 SKIPJACK-PRIVATE-WRAP

Механизм SKIPJACK-PRIVATE-WRAP, обозначенный как **CKM_SKIPJACK_PRIVATE_WRAP**, используется для упаковки и распаковки секретного ключа MEK. Он может упаковывать и распаковывать секретные ключи KEA и DSA.

Данный механизм имеет параметр структуры **CK_SKIPJACK_PRIVATE_WRAP_PARAMS**.

12.16.14 SKIPJACK-RELAYX

Механизм SKIPJACK-RELAYX, обозначенный как **CKM_SKIPJACK_RELAYX**, ипользуется с функцией **C_WrapKey,** для того чтобы «изменить упаковку» на секретном ключе, который был упакован при помощи механизма SKIPJACK-PRIVATE-WRAP (см. 12.16.13).

Данный механизм имеет параметр структуры **CK_SKIPJACK_RELAYX_PARAMS**.

Хотя механизм SKIPJACK-RELAYX используется с **C_WrapKey**, он отличается от других механизмов упаковки ключа. Другие механизмы упаковки ключа принимают управление ключом (key handle) как один из параметров (arguments) для **C_WrapKey**; однако для механизма SKIPJACK_RELAYX [всегда недопустимое] значение 0 должно быть передано как параметр управления ключом в **C_WrapKey**, а уже упакованный ключ должен быть передан как часть структуры **CK_SKIPJACK_RELAYX_PARAMS**.

12.17 BATON

12.17.1 Определения

В настоящем разделе тип ключа CKK_BATON определяется как CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевого объекта.

Механизмы:
CKM_BATON_KEY_GEN
CKM_BATON_ECB128
CKM_BATON_ECB96
CKM_BATON_CBC128
CKM_BATON_COUNTER
CKM_BATON_SHUFFLE
CKM_BATON_WRAP

12.17.2 Объекты секретного ключа BATON

Объекты секретного ключа BATON (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_BATON**) содержат ключи BATON единой длины. В таблице 115 определены атрибуты объекта секретного ключа BATON в дополнение к общим атрибутам, определенным для объекта данного класса:

Таблица 115 – Объект секретного ключа BATON

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (всегда длиной в 40 байт)
См. сноски в таблице 15.		

Ключи BATON имеют 160 бит контрольной суммы, и эти биты должны быть верно размещены. Попытка создать или распаковать ключ BATON с неверной контрольной суммой приведет к выдаче сообщения об ошибке.

Не ясно, существует ли (и будет ли вообще существовать) носитель, который позволит своему программному приложению создать ключ BATON с заданным значением. Тем не менее приводим шаблоны, предназначенные для этой цели.

Ниже приведен шаблон для создания объекта секретного ключа BATON MEK:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_BATON;
CK_UTF8CHAR label[] = “Объект секретного ключа BATON MEK”;
CK_BYTE value[40] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

Далее приведен шаблон для создания объекта секретного ключа BATON TEK:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_BATON;
CK_UTF8CHAR label[] = “Объект секретного ключа BATON TEK”;
CK_BYTE value[40] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},

  {CKA_WRAP, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.17.3 Генерация ключа BATON

Механизм генерации ключа BATON, именуемый **CKM_BATON_KEY_GEN**, является механизмом генерации ключа для BATON. На выходе данного механизма получается величина, именуемая ключ шифрования сообщения (Message Encryption Key, MEK).

Данный механизм не имеет параметра.

Данный механизм добавляет к новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE**.

12.17.4 BATON-ECB128

Механизм BATON-ECB128, обозначенный как **CKM_BATON_ECB128**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением BATON в режиме 128-битной электронной кодовой книги.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 116.

Таблица 116 – BATON-ECB128: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	BATON	Кратна 16	Та же, что и длина ввода	Без заключительной части
C_Decrypt	BATON	Кратна 16	Та же, что и длина ввода	Без заключительной части

12.17.5 BATON-ECB96

Механизм BATON-ECB96, обозначенный как **CKM_BATON_ECB96**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением BATON в режиме 96-битной электронной кодовой книги.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 117.

Таблица 117 – BATON-ECB96: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	BATON	Кратна 12	Та же, что и длина ввода	Без заключительной части
C_Decrypt	BATON	Кратна 12	Та же, что и длина ввода	Без заключительной части

12.17.6 BATON-CBC128

Механизм BATON-CBC128, обозначенный как **CKM_BATON_CBC128**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением BATON в 128-битном режиме цепочки шифрблоков.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 118.

Таблица 118 – BATON-CBC128: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	BATON	Кратна 16	Та же, что и длина ввода	Без заключительной части
C_Decrypt	BATON	Кратна 16	Та же, что и длина ввода	Без заключительной части

12.17.7 BATON-COUNTER

Механизм BATON-COUNTER, обозначенный как **CKM_BATON_COUNTER**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением BATON в режиме счетчика.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 119.

Таблица 119 – BATON-COUNTER: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	BATON	Кратна 16	Та же, что и длина ввода	Без заключительной части
C_Decrypt	BATON	Кратна 16	Та же, что и длина ввода	Без заключительной части

12.17.8 BATON-SHUFFLE

Механизм BATON-SHUFFLE, обозначенный как **CKM_BATON_SHUFFLE**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением BATON в режиме сдвига.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 120.

Таблица 120 – BATON-SHUFFLE: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	BATON	Кратна 16	Та же, что и длина ввода	Без заключительной части
C_Decrypt	BATON	Кратна 16	Та же, что и длина ввода	Без заключительной части

12.17.9 BATON WRAP

Механизм упаковки и распаковки BATON, обозначенный как **CKM_BATON_WRAP**, является функцией, используемой для упаковки и распаковки секретного ключа (MEK). Он может упаковывать и распаковывать ключи SKIPJACK, BATON и JUNIPER.

Данный механизм не имеет параметров.

При использовании для распаковки ключа данный механизм добавляет в него атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE**.

12.18 JUNIPER

12.18.1 Определения

В настоящем разделе тип ключа CKK_JUNIPER определяется как CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевого объекта.

Механизмы:

```
CKM_JUNIPER_KEY_GEN
CKM_JUNIPER_ECB128
CKM_JUNIPER_CBC128
CKM_JUNIPER_COUNTER
CKM_JUNIPER_SHUFFLE
CKM_JUNIPER_WRAP
```

12.18.2 Объекты секретного ключа JUNIPER

Объекты секретного ключа JUNIPER (класс объекта – **CKO_SECRET_KEY,** тип ключа – **CKK_ JUNIPER**) содержат ключи JUNIPER единой длины. В таблице 121 определены атрибуты объекта секретного ключа JUNIPER в дополнение к общим атрибутам, определенным для объекта данного класса.

Таблица 121 – Объект секретного ключа JUNIPER

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа (всегда длиной в 40 байт)
См. сноски в таблице 15.		

Ключи JUNIPER имеют 160 бит контрольной суммы, и эти биты должны быть верно размещены. Попытка создать или распаковать ключ JUNIPER с неверной контрольной суммой приведут к выдаче сообщения об ошибке.

Не ясно, существует ли (и будет ли вообще существовать) носитель, который позволит своему программному приложению создать ключ JUNIPER с заданным значением. Тем не менее приводим шаблоны, предназначенные для этой цели.

Ниже приведен шаблон для создания объекта секретного ключа JUNIPER MEK:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_JUNIPER;
CK_UTF8CHAR label[] = "Объект секретного ключа JUNIPER MEK";
CK_BYTE value[40] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

Далее приведен шаблон для создания объекта секретного ключа JUNIPER TEK:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_JUNIPER;
CK_UTF8CHAR label[] = “Объект секретного ключа JUNIPER TEK”;
CK_BYTE value[40] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_WRAP, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.18.3 Генерация ключа JUNIPER

Механизм генерации ключа JUNIPER, именуемый **CKM_JUNIPER_KEY_GEN**, является механизмом генерации ключа для JUNIPER. На выходе данного механизма получается величина, именуемая ключ шифрования сообщения (Message Encryption Key, MEK).

Данный механизм не имеет параметра.

Данный механизм добавляет к новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE**.

12.18.4 JUNIPER-ECB128

Механизм JUNIPER-ECB128, обозначенный как **CKM_JUNIPER_ECB128**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением JUNIPER в режиме 128-битной электронной кодовой книги.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 122. Данные (части) ввода и вывода для шифрования и расшифровки могут начинаться с одной и той же позиции в памяти.

Таблица 122 – JUNIPER-ECB128: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	JUNIPER	Кратна 16	Та же, что и длина ввода	Без заключительной части
C_Decrypt	JUNIPER	Кратна 16	Та же, что и длина ввода	Без заключительной части

12.18.5 JUNIPER-CBC128

Механизм JUNIPER-CBC128, обозначенный как **CKM_JUNIPER_CBC128**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением JUNIPER в 128-битном режиме цепочки шифрблоков.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 123. Данные (части) ввода и вывода для шифрования и расшифровки могут начинаться с одной и той же позиции в памяти.

Таблица 123 – JUNIPER-CBC128: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	JUNIPER	Кратна 16	Та же, что и длина ввода	Без заключительной части
C_Decrypt	JUNIPER	Кратна 16	Та же, что и длина ввода	Без заключительной части

12.18.6 JUNIPER-COUNTER

Механизм JUNIPER-COUNTER, обозначенный как **CKM_JUNIPER_COUNTER**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением JUNIPER в режиме счетчика.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 124. Данные (части) ввода и вывода для шифрования и расшифровки могут начинаться с одной и той же позиции в памяти.

Таблица 124 – JUNIPER-COUNTER: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	JUNIPER	Кратна 16	Та же, что и длина ввода	Без заключительной части
C_Decrypt	JUNIPER	Кратна 16	Та же, что и длина ввода	Без заключительной части

12.18.7 JUNIPER-SHUFFLE

Механизм JUNIPER-SHUFFLE, обозначенный как **CKM_JUNIPER_SHUFFLE**, является механизмом для одностороннего и многостороннего шифрования и расшифровки с применением JUNIPER в режиме сдвига.

Данный механизм имеет параметр – 24-байтный вектор инициализации IV. В ходе операции шифрования этот вектор инициализации IV задается с определенным значением, сгенерированным носителем, т. е. данное приложение не может задать конкретный вектор инициализации IV при шифровании. Данное приложение может задать конкретный вектор инициализации IV при расшифровке.

Ограничения на типы ключей и длину данных приведены в таблице 125. Данные (части) ввода и вывода для шифрования и расшифровки могут начинаться с одной и той же позиции в памяти.

Таблица 125 – JUNIPER-SHUFFLE: Данные и длина

Функция	Тип ключа	Длина ввода	Длина вывода	Примечания
C_Encrypt	JUNIPER	Кратна 16	Та же, что и длина ввода	Без заключительной части
C_Decrypt	JUNIPER	Кратна 16	Та же, что и длина ввода	Без заключительной части

12.18.8 JUNIPER WRAP

Механизм упаковки и распаковки JUNIPER, обозначенный как **CKM_JUNIPER_WRAP**, является функцией, используемой для упаковки и распаковки секретного ключа MEK. Он может упаковывать и распаковывать ключи SKIPJACK, BATON и JUNIPER.

Данный механизм не имеет параметров.

При использовании для распаковки ключа данный механизм добавляет в него атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE**.

12.19 MD2

12.19.1 Определения

Механизмы:

```
CKM_MD2
CKM_MD2_HMAC
CKM_MD2_HMAC_GENERAL
CKM_MD2_KEY_DERIVATION
```

12.19.2 Механизм алгоритма списка сообщения MD2

Механизм MD2, обозначенный как **CKM_MD2**, представляет собой механизм получения краткого содержания (списка) сообщения (Message Digest, MD) посредством реализации алгоритма, описанного в RFC 1319.

Данный механизм не имеет параметра.

Ограничения на длину данных приведены в таблице 126.

Таблица 126 – MD2: Длина данных

Функция	Длина данных	Длина краткого содержания (списка)
C_Digest	Любая	16

12.19.3 Механизм общей длины MD2-HMAC

Механизм общей длины MD2-HMAC, обозначенный как **CKM_MD2_HMAC_GENERAL**, представляет собой механизм для постановки подписей и верификации. Данный механизм использует конструкцию HMAC, основанную на функции хэширования MD2. Используемые данным механизмом ключи являются общими секретными ключами.

Данный механизм имеет параметр **CK_MAC_GENERAL_PARAMS**, который содержит длину требуемого выводного массива в байтах. Эта длина должна быть в диапазоне (длина вывода MD2 – 16 байт). Подписи (MAC), производимые данным механизмом, будут браться из начала полного 16-байтного выводного массива HMAC.

Таблица 127 – Механизм общей длины MD2-HMAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	Общий секретный	Любая	0 – 16, в зависимости от параметров
C_Verify	Общий секретный	Любая	0 – 16, в зависимости от параметров

12.19.4 MD2-HMAC

Механизм MD2-HMAC, обозначенный как **CKM_MD2_HMAC**, представляет собой частный случай механизма общей длины MD2-HMAC, описанного в 12.19.3.

Данный механизм не имеет параметра и всегда производит выводной массив длиной 16 байт.

12.19.5 Деривация ключа MD2

Деривация ключа MD2, обозначенная как **CKM_MD2_KEY_DERIVATION**, является механизмом, предоставляющим возможность получения секретного ключа посредством получения списка значения другого секретного ключа при помощи MD2.

Вначале вычисляется список значения базового ключа, после чего этот результат используется для получения значения выводимого секретного ключа.

- Если шаблон не предоставляет длину или тип ключа, тогда ключ, произведенный данным механизмом, будет общим секретным ключом. Его длина будет 16 байт (размер вывода MD2).
- Если шаблон не предоставляет тип ключа, но предоставляет длину, тогда ключ, произведенный данным механизмом, будет общим секретным ключом заданной длины.
- Если шаблон не предоставляет длину ключа, но предоставляет тип ключа, тогда этот ключ должен иметь верно заданную длину. Если он ее имеет, тогда ключ, произведенный данным механизмом, будет

того типа, который задан в шаблоне. Если это не так, будет выдано сообщение об ошибке.

- Если шаблон предоставляет и тип, и длину ключа, то длина должна быть совместима с типом ключа. Ключ, произведенный данным механизмом, будет иметь заданные тип и длину.

Если при помощи этого механизма выводится ключ DES, DES2 или CDMF, биты четности данного ключа должны быть верно расположены.

Если заданный тип ключа требует более чем 16 байт (например, DES3), генерируется сообщение об ошибке.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE.**

12.20 MD5

12.20.1 Определения

Механизмы:
CKM_MD5
CKM_MD5_HMAC
CKM_MD5_HMAC_GENERAL
CKM_MD5_KEY_DERIVATION

12.20.2 Список MD5

Механизм MD5, обозначенный как **CKM_MD5**, представляет собой механизм получения краткого содержания (списка) сообщения (Message Digest, MD) посредством реализации алгоритма, описанного в RFC 1319.

Данный механизм не имеет параметра.

Ограничения на длину данных приведены в таблице 128.

Таблица 128 – MD5: Длина данных

Функция	Длина данных	Длина краткого содержания (списка)
C_Digest	Любая	16

12.20.3 Механизм общей длины MD5-HMAC

Механизм общей длины MD5-HMAC, обозначенный как **CKM_MD5_HMAC_GENERAL**, представляет собой механизм для постановки подписей и верификации. Данный механизм использует конструкцию HMAC, основанную на функции хэширования MD5. Используемые данным механизмом ключи являются общими секретными ключами.

Данный механизм имеет параметр **CK_MAC_GENERAL_PARAMS**, который содержит длину требуемого выводного массива в байтах. Эта длина должна быть в диапазоне 0 – 16 (длина вывода MD5 – 16 байт). Подписи (MAC), производимые данным механизмом, будут браться из начала полного 16-байтного выводного массива HMAC.

Таблица 129 – Механизм общей длины MD5-HMAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	Общий секретный	Любая	0 – 16, в зависимости от параметров
C_Verify	Общий секретный	Любая	0 – 16, в зависимости от параметров

12.20.4 MD5-HMAC

Механизм MD5-HMAC, обозначенный как **CKM_MD5_HMAC**, представляет собой частный случай механизма общей длины MD5-HMAC, описанного в 12.20.3.

Данный механизм не имеет параметра и всегда производит выводной массив длиной 16 байт.

12.20.5 Деривация ключей MD5

Деривация ключа MD5, обозначенная как **CKM_MD5_KEY_DERIVATION**, является механизмом, предоставляющим возможность получения секретного ключа посредством получения списка значения другого секретного ключа при помощи MD5.

Вначале вычисляется список значения базового ключа, после чего этот результат используется для получения значения выводимого секретного ключа.

- Если шаблон не предоставляет длину или тип ключа, тогда ключ, произведенный данным механизмом, будет общим секретным ключом. Его длина будет 16 байт (размер вывода MD5).
- Если шаблон не предоставляет тип ключа, но предоставляет длину, тогда ключ, произведенный данным механизмом, будет общим секретным ключом заданной длины.
- Если шаблон не предоставляет длину ключа, но предоставляет тип ключа, тогда этот ключ должен иметь верно заданную длину. Если он ее имеет, тогда ключ, произведенный данным механизмом, будет того типа, который задан в шаблоне. Если это не так, будет выдано сообщение об ошибке.
- Если шаблон предоставляет и тип, и длину ключа, то длина должна быть совместима с типом ключа. Ключ, произведенный данным механизмом, будет иметь заданные тип и длину.

Если при помощи этого механизма выводится ключ DES, DES2 или CDMF, биты четности данного ключа должны быть расположены верно.

Если заданный тип ключа требует более чем 16 байт (например, DES3), генерируется сообщение

об ошибке.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

12.21 SHA-1

12.21.1 Определения

Механизмы:
`CKM_SHA_1`
`CKM_SHA_1_HMAC`
`CKM_SHA_1_HMAC_GENERAL`
`CKM_SHA1_KEY_DERIVATION`

12.21.2 Список SHA-1

Механизм SHA-1, обозначенный как **CKM_SHA_1**, представляет собой механизм для получения списка сообщения посредством применения алгоритма хэширования Secure Hash Algorithm (SHA) со 160-битным списком сообщения, как описано в FIPS PUB 180-2.

Данный механизм не имеет параметра.

Ограничения на длину данных ввода и вывода приведены в таблице 130. Для одностороннего осуществления списка данные и список могут начинаться с одной и той же позиции в памяти.

Таблица 130 – SHA-1: Длина данных

Функция	Длина ввода	Длина списка
C_Digest	Любая	20

12.21.3 Механизм общей длины SHA-1-HMAC

Механизм общей длины SHA-1-HMAC, обозначенный как **CKM_SHA_1_HMAC_GENERAL**, представляет собой механизм для постановки подписей и верификации. Данный механизм использует конструкцию HMAC, основанную на функции хэширования SHA-1. Используемые данным механизмом ключи являются общими секретными ключами.

Данный механизм имеет параметр **CK_MAC_GENERAL_PARAMS**, который содержит длину требуемого выводного массива в байтах. Эта длина должна быть в диапазоне 0 – 20 (длина вывода MD5 – 16 байт). Подписи (MAC), производимые данным механизмом, будут браться из начала полного 20-байтного выводного массива HMAC.

Таблица 131 – Механизм общей длины SHA-1-HMAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	Общий секретный	Любая	0 – 20, в зависимости от параметров
C_Verify	Общий секретный	Любая	0 – 20, в зависимости от параметров

12.21.4 SHA-1-HMAC

Механизм SHA-1-HMAC, обозначенный как **CKM_SHA_1_HMAC**, является частным случаем механизма общей длины SHA-1-HMAC, описанного в 12.21.3.

Данный механизм не имеет параметра и всегда производит выводной массив длиной 20 байт.

12.21.5 Деривация ключа SHA-1

Деривация ключа SHA-1, обозначенная как **CKM_SHA1_KEY_DERIVATION**, является механизмом,

предоставляющим возможность получения секретного ключа посредством получения списка значения другого секретного ключа при помощи SHA-1.

Вначале вычисляется список значения базового ключа, после чего этот результат используется для получения значения выводимого секретного ключа.

- Если шаблон не предоставляет тип ключа, но предоставляет длину, тогда ключ, произведенный данным механизмом, будет общим секретным ключом заданной длины. Его длина будет 20 байт (размер вывода SHA-1).
- Если шаблон не предоставляет тип ключа, но предоставляет длину, тогда ключ, произведенный данным механизмом, будет общим секретным ключом заданной длины.
- Если шаблон не предоставляет длину ключа, но предоставляет тип ключа, тогда этот ключ должен иметь верно заданную длину. Если он ее имеет, тогда ключ, произведенный данным механизмом, будет того типа, который задан в шаблоне. Если это не так, будет выдано сообщение об ошибке.
- Если шаблон предоставляет и тип, и длину ключа, то длина должна быть совместима с типом ключа. Ключ, произведенный данным механизмом, будет иметь заданные тип и длину.

Если при помощи этого механизма выводится ключ DES, DES2 или CDMF, биты четности данного ключа должны быть расположены верно.

Если заданный тип ключа требует более чем 20 байт (например, DES3), генерируется сообщение об ошибке.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

12.22 SHA-256

12.22.1 Определения

Механизмы:
`CKM_SHA256`
`CKM_SHA256_HMAC`
`CKM_SHA256_HMAC_GENERAL`
`CKM_SHA256_KEY_DERIVATION`

12.22.2 Список SHA-256

Механизм SHA-256, обозначенный как **CKM_SHA_256**, представляет собой механизм для получения списка сообщения посредством применения алгоритма хэширования Secure Hash Algorithm (SHA) со 256-битным списком сообщения, как описано в FIPS PUB 180-2.

Данный механизм не имеет параметра.

Ограничения на длину данных ввода и вывода приведены в таблице 132. Для одностороннего осуществления списка, данные и список могут начинаться с одной и той же позиции в памяти.

Таблица 132 – SHA-1: Длина данных

Функция	Длина ввода	Длина списка
C_Digest	Любая	32

12.22.3 Механизм общей длины SHA-256-HMAC

Механизм общей длины SHA-256-HMAC, обозначенный как **CKM_SHA_256_HMAC_GENERAL**, представляет собой механизм, аналогичный механизму общей длины SHA-1-HMAC, описанному в 12.21.3, за исключением того, что он использует конструкцию HMAC, основанную на функции хэширования SHA-256, и длина его вывода должна быть в диапазоне 0 – 32. Используемые данным механизмом ключи являются общими секретными ключами, которые должны быть как минимум 16-байтными, что составляет половину размера выходного массива хэш-алгоритма SHA-256.

Данный механизм имеет параметр **CK_MAC_GENERAL_PARAMS**, который содержит длину требуемого выводного массива в байтах. Эта длина должна быть в диапазоне 0 – 32 (длина вывода SHA-256 – 32 байта). Носители в соответствии с документом HMAC FIPS-198 могут ограничивать длину вывода минимальным значением в 4 или 16 (половина максимальной длины). Подписи (MAC), производимые данным механизмом, будут браться из начала полного 32-байтного выводного массива HMAC.

Таблица 133 – Механизм общей длины SHA-256-HMAC: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	Общий секретный	Любая	0 – 32, в зависимости от параметров
C_Verify	Общий секретный	Любая	0 – 32, в зависимости от параметров

12.22.4 SHA-256-HMAC

Механизм SHA-256-HMAC, обозначенный как **CKM_SHA_256_HMAC**, является частным случаем механизма общей длины SHA-256-HMAC, описанного в 12.22.3.

Данный механизм не имеет параметра и всегда производит выводной массив длиной 32.

12.22.5 Деривация ключа SHA-256

Механизм деривации ключа SHA-256, обозначенный как **CKM_SHA256_KEY_DERIVATION**, аналогичен механизму деривации ключа SHA-1, описанному в 12.21.5, за исключением того, что используется функция хэширования SHA-256 и значимая длина составляет 32 байта.

12.23 SHA-384

12.23.1 Определения

Механизмы:
`CKM_SHA384`
`CKM_SHA384_HMAC`
`CKM_SHA384_HMAC_GENERAL`
`CKM_SHA384_KEY_DERIVATION`

12.23.2 Список SHA-384

Механизм SHA-384, обозначенный как **CKM_SHA384**, представляет собой механизм для получения списка сообщения посредством применения алгоритма хэширования Secure Hash Algorithm (SHA) с 384-битным списком сообщения, как описано в FIPS PUB 180-2.

Данный механизм не имеет параметра.

Ограничения на длину данных ввода и вывода приведены в таблице 134. Для одностороннего осуществления списка данные и список могут начинаться с одной и той же позиции в памяти.

Таблица 134 – SHA-384: Длина данных

Функция	Длина ввода	Длина списка
C_Digest	Любая	48

12.23.3 Механизм общей длины SHA-384-HMAC

Механизм общей длины SHA-384-HMAC, обозначенный как **CKM_SHA_384_HMAC_GENERAL**, представляет собой механизм, аналогичный механизму общей длины SHA-1-HMAC, описанному в 12.21.3, за исключением того, что он использует конструкцию HMAC, основанную на функции хэширования SHA-384, и длина его вывода должна быть в диапазоне 0 – 48.

12.23.4 SHA-384-HMAC

Механизм SHA-384-HMAC, обозначенный как **CKM_SHA384_HMAC**, является частным случаем механизма общей длины SHA-384-HMAC.

Данный механизм не имеет параметра и всегда производит выводной массив длиной 48 байт.

12.23.5 Деривация ключа SHA-384

Механизм деривации ключа SHA-384, обозначенный как **CKM_SHA384_KEY_DERIVATION**, аналогичен механизму деривации ключа SHA-1, описанному в 12.21.5, за исключением того, что используется функция хэширования SHA-384 и значимая длина составляет 48 байт.

12.24 SHA-512

12.24.1 Определения

Механизмы:

```
CKM_SHA512
CKM_SHA512_HMAC
CKM_SHA512_HMAC_GENERAL
CKM_SHA512_KEY_DERIVATION
```

12.24.2 Список SHA-512

Механизм SHA-512, обозначенный как **CKM_SHA512**, представляет собой механизм для получения списка сообщения посредством применения алгоритма хэширования Secure Hash Algorithm (SHA) с 512-битным списком сообщения, как описано в FIPS PUB 180-2.

Данный механизм не имеет параметра.

Ограничения на длину данных ввода и вывода приведены в таблице 135. Для одностороннего осуществления списка данные и список могут начинаться с одной и той же позиции в памяти.

Таблица 135 – SHA-512: Длина данных

Функция	Длина ввода	Длина списка
C_Digest	Любая	64

12.24.3 Механизм общей длины SHA-512-HMAC

Механизм общей длины SHA-512-HMAC, обозначенный как **CKM_SHA512_HMAC_GENERAL**, представляет собой механизм, аналогичный механизму общей длины SHA-1-HMAC, описанному в 12.21.3, за исключением того, что он использует конструкцию HMAC, основанную на функции хэширования SHA-512, и длина его вывода должна быть в диапазоне 0 – 64.

12.24.4 SHA-512-HMAC

Механизм SHA-512-HMAC, обозначенный как **CKM_SHA512_HMAC**, является частным случаем механизма общей длины SHA-512-HMAC.

Данный механизм не имеет параметра и всегда производит выводной массив длиной 64 байт.

12.24.5 Деривация ключа SHA-512

Механизм деривации ключа SHA-512, обозначенный как **CKM_SHA512_KEY_DERIVATION**, аналогичен механизму деривации ключа SHA-1, описанному в 12.21.5, за исключением того, что используется функция хэширования SHA-512 и значимая длина составляет 64 байта.

12.25 FASTHASH

12.25.1 Определения

Механизмы:

```
CKM_FASTHASH
```

12.25.2 Список FASTHASH

Механизм FASTHASH, обозначенный как **CKM_FASTHASH**, представляет собой механизм для получения списка сообщения по алгоритму правительства США.

Данный механизм не имеет параметра.

Ограничения на длину данных ввода и вывода приведены в таблице 136.

Таблица 136 – FASTHASH: Длина данных

Функция	Длина ввода	Длина списка
C_Digest	Любая	40

12.26 Шифрование на основе пароля на основании PKCS #5 и аналогично требованиям PKCS #5

Механизмы, описываемые в данной секции, предназначены для генерации ключей и векторов инициализации (IV) с применением шифрования на основе пароля (password-based encryption, PBE). Данный метод, используемый для генерации ключей и вектора инициализации IV, определен в PKCS #5.

12.26.1 Определения

Механизмы:

```
CKM_PBE_MD2_DES_CBC
CKM_PBE_MD5_DES_CBC
CKM_PBE_MD5_CAST_CBC
CKM_PBE_MD5_CAST3_CBC
CKM_PBE_MD5_CAST5_CBC
CKM_PBE_MD5_CAST128_CBC
CKM_PBE_SHA1_CAST5_CBC
CKM_PBE_SHA1_CAST128_CBC
CKM_PBE_SHA1_RC4_128
CKM_PBE_SHA1_RC4_40
CKM_PBE_SHA1_DES3_EDE_CBC
CKM_PBE_SHA1_DES2_EDE_CBC
CKM_PBE_SHA1_RC2_128_CBC
CKM_PBE_SHA1_RC2_40_CBC
CKM_PKCS5_PBKD2
CKM_PBA_SHA1_WITH_SHA1_HMAC
```

12.26.2 Параметры механизма шифрования/аутентификации на основе пароля

♦ **CK_PBE_PARAMS; CK_PBE_PARAMS_PTR**

CK_PBE_PARAMS является структурой, предоставляющей всю необходимую информацию, которая требуется для механизмов CKM_PBE и CKM_PBA_SHA1_WITH_SHA1_HMAC (для получения информации по механизмам генерации на основе PBE (см. PKCS #5 и PKCS #12). Данная структура задается следующим образом:

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

Поля данной структуры имеют следующие значения:

pInitVector	указатель на место, куда поступает 8-байтный вектор инициализации (IV), если требуется вектор инициализации IV;
pPassword	точки к паролю, который предстоит использовать в генерации ключа методом PBE;
ulPasswordLen	длина парольной информации в байтах;
pSalt	точки к случайной величине «salt», которую предстоит использовать при генерации ключа методом PBE;
ulSaltLen	длина в байтах информации «salt»;
ulIteration	число итераций, требуемых для генерации.

CK_PBE_PARAMS_PTR является указателем на **CK_PBE_PARAMS**.

12.26.3 MD2-PBE для DES-CBC

Механизм MD2-PBE для DES-CBC, обозначенный как **CKM_PBE_MD2_DES_CBC**, представляет собой механизм, используемый для генерации секретного ключа DES и вектора инициализации IV из пароля и величины «salt» посредством алгоритма списка MD2 и счетчика итераций. Эта функциональность определена в PKCS #5 как PBKDF1.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

12.26.4 MD5-PBE для DES-CBC

Механизм MD5-PBE для DES-CBC, обозначенный как **CKM_PBE_MD5_DES_CBC**, представляет собой механизм, используемый для генерации секретного ключа DES и вектора инициализации IV из пароля и величины «salt» посредством алгоритма списка MD5 и счетчика итераций. Эта функциональность определена в PKCS #5 как PBKDF1.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает инфор-

мацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

12.26.5 MD5-PBE для CAST-CBC

Механизм MD5-PBE для CAST-CBC, обозначенный как **CKM_PBE_MD5_CAST_CBC**, представляет собой механизм, используемый для генерации секретного ключа CAST и вектора инициализации IV из пароля и величины «salt» посредством алгоритма списка MD5 и счетчика итераций. Функциональность данного механизма аналогична той, что определена в PKCS #5 как PBKDF1 для MD5 и DES.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

Длина ключа CAST, генерируемого данным механизмом, может быть определена в прилагаемом шаблоне; если в шаблоне ее нет, то ее значение по умолчанию будет 8 байт.

12.26.6 MD5-PBE для CAST3-CBC

Механизм MD5-PBE для CAST3-CBC, обозначенный как **CKM_PBE_MD5_CAST3_CBC**, представляет собой механизм, используемый для генерации секретного ключа CAST3 и IV из пароля и величины «salt» посредством алгоритма списка MD5 и счетчика итераций. Функциональность данного механизма аналогична той, что определена в PKCS #5 как PBKDF1 для MD5 и DES.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

Длина ключа CAST3, генерируемого данным механизмом, может быть определена в прилагаемом шаблоне; если в шаблоне ее нет, то ее значение по умолчанию будет 8 байт.

12.26.7 MD5-PBE для CAST128-CBC (CAST5-CBC)

Механизм MD5-PBE для CAST128-CBC (CAST5-CBC), обозначенный как **CKM_PBE_MD5_CAST128_CBC** или **CKM_PBE_MD5_CAST5_CBC**, представляет собой механизм, используемый для генерации секретного ключа CAST128-CBC (CAST5-CBC) и вектора инициализации IV из пароля и величины «salt» посредством алгоритма списка MD5 и счетчика итераций. Функциональность данного механизма аналогична той, что определена в PKCS #5 как PBKDF1 для MD5 и DES.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

Длина ключа CAST128 (CAST5), генерируемого данным механизмом, может быть определена в прилагаемом шаблоне; если в шаблоне ее нет, то ее значение по умолчанию будет 8 байт.

12.26.8 SHA-1-PBE для CAST128-CBC (CAST5-CBC)

Механизм SHA-1-PBE для CAST128-CBC (CAST5-CBC), обозначенный как **CKM_PBE_SHA1_CAST128_CBC** или **CKM_PBE_SHA1_CAST5_CBC**, представляет собой механизм, используемый для генерации секретного ключа CAST128-CBC (CAST5-CBC) и вектора инициализации IV из пароля и величины «salt» посредством алгоритма списка SHA1 и счетчика итераций. Функциональность данного механизма аналогична той, что определена в PKCS #5 как PBKDF1 для MD5 и DES.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

Длина ключа CAST128 (CAST5), генерируемого данным механизмом, может быть определена в прилагаемом шаблоне; если в шаблоне ее нет, то ее значение по умолчанию будет 8 байт.

12.26.9 Параметры механизма генерации ключа PBKDF2 в соответствии с PKCS #5

♦ **CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE; CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE_PTR**

Параметр **CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE** используется для задания псевдослучайной функции (Pseudo-Random Function, PRF), применяемой для генерации ключевых бит с использованием PKCS #5 PBKDF2. Данный параметр задается следующим образом:

```
typedef CK_ULONG CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE;
```

Применяемые затем PRF определены в PKCS #5 v2.0. Данные функции перечислены в таблице 137.

Таблица 137 – Генерация ключа PKCS #5 PBKDF2: Псевдослучайные функции

Идентификатор источника	Значение	Тип параметра
CKP_PKCS5_PBKD2_HMAC_SHA1	0x00000001	Параметр отсутствует. Поле *pPrfData* должно быть NULL, а поле *ulPrfDataLen* должно быть нулем

CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE_PTR является указателем на **CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE**.

♦ **CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE; CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE_PTR**

Параметр **CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE** используется для обозначения источника величины «salt» при деривации ключа с использованием PKCS #5 PBKDF2. Данный параметр задается следующим образом:

```
typedef CK_ULONG CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE;
```

В PKCS #5 v2.0 указаны приведенные ниже источники величины «salt». В таблице 138 указанные источники приведены вместе с соответствующим типом данных для поля *pSaltSourceData* в описываемой ниже структуре **CK_PKCS5_PBKD2_PARAM.**

Таблица 138 – Генерация ключа PKCS #5 PBKDF2: Источники величины «salt»

Идентификатор источника	Значение	Тип данных
CKZ_SALT_SPECIFIED	0x00000001	Совокупность CK_BYTE, содержащая значение величины «salt»

CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE_PTR является указателем на **CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE**.

♦ **CK_ PKCS5_PBKD2_PARAMS; CK_PKCS5_PBKD2_PARAMS_PTR**

CK_PKCS5_PBKD2_PARAMS представляет собой структуру, предоставляющую параметры механизму **CKM_PKCS5_PBKD2**. Данная структура задается следующим образом:

```
typedef struct CK_PKCS5_PBKD2_PARAMS {
  CK_PKCS5_PBKDF2_SALT_SOURCE_TYPE saltSource;
  CK_VOID_PTR pSaltSourceData;
  CK_ULONG ulSaltSourceDataLen;
  CK_ULONG iterations;
  CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf;
  CK_VOID_PTR pPrfData;
  CK_ULONG ulPrfDataLen;  CK_UTF8CHAR_PTR pPassword;
  CK_ULONG_PTR ulPasswordLen;
} CK_PKCS5_PBKD2_PARAMS;
```

Поля данной структуры имеют следующие значения:

saltSource — источник величины «salt»;
pSaltSourceData — данные, используемые как ввод для источника «salt»;
ulSaltSourceDataLen — длина ввода источника «salt»;
iterations — количество итераций, которые надлежит выполнить при генерации каждого блока случайных данных;
prf — псевдослучайная функция, которую надлежит использовать для генерации ключа;
pPrfData — данные, используемые как ввод для псевдослучайной функции в дополнение к величине «salt»;
ulPrfDataLen — длина ввода данных для псевдослучайной функции;

pPassword точки для пароля, который должен быть использован при генерации ключа методом PBE;
ulPasswordLen длина парольной информации в байтах.

CK_PKCS5_PBKD2_PARAMS_PTR является указателем на **CK_PKCS5_PBKD2_PARAMS**.

12.26.10 Генерация ключа PKCS #5 PBKD2

Генерация ключа PKCS #5 PBKDF2, именуемая **CKM_PKCS5_PBKD2**, представляет собой механизм, используемый для генерации секретного ключа из пароля и величины «salt». Данная функциональность определена в PKCS #5 как PBKDF2.

Данный механизм имеет параметр структуры **CK_PKCS5_PBKD2_PARAMS**. Данный параметр указывает источник величины «salt», псевдослучайную функцию и число итераций, применяемых для генерации нового ключа.

Поскольку данный механизм может быть использован для генерации любого типа секретного ключа, шаблоны нового ключа должны содержать атрибуты **CKA_KEY_TYPE** и **CKA_VALUE_LEN**. Если тип ключа имеет фиксированную длину, атрибут **CKA_VALUE_LEN** может опускаться.

12.27 Механизмы шифрования/аутентификации на основе пароля согласно PKCS #12

Механизмы, описываемые в данном разделе, предназначены для генерации ключей и векторов инициализации IV для выполнения шифрования или аутентификации на основе пароля. Метод, применяемый для генерации ключей и вектора инициализации IV, основан на методике, конкретизированной в PKCS #12.

Здесь мы детально описываем общий метод выработки различных типов псевдослучайных бит из пароля, p, битовой строки «salt», s, а также отсчета итераций, c. Тип псевдослучайных бит, которые предстоит вырабатывать, определяется идентификационным байтом, ID, значение которого будет установлено позже.

Пусть H – функция хэширования, построенная вокруг функции сжатия $f: \boldsymbol{Z}_2^u \times \boldsymbol{Z}_2^v \to \boldsymbol{Z}_2^u$ (т. е. H имеет цепочную переменную и вывод длиной u бит, а также ввод сообщения в функцию сжатия H – v bits). Для MD2 и MD5 $u = 128$ и $v = 512$; для SHA-1 $u = 160$ и $v = 512$.

Принимаем для данного случая, что u и v оба являются кратными 8, как и длины в битах паролей и строк «salt», а также количество n требуемых псевдослучайных бит. И, конечно, u и v не равны нулю.

1. Создаем строку D (так называемый диверсификатор) посредством соединения $v/8$ копий ID.

2. Соединяем вместе копии «salt» для создания строки S длиной $v \cdot \lceil s/v \rceil$ бит (для создания S замыкающая копия «salt» может быть усечена). Отметим, что если «salt» будет пустой строкой, то таким же будет и S.

3. Соединяем вместе копии пароля для создания строки P длиной $v \cdot \lceil p/v \rceil$ бит (для создания P замыкающая копия пароля может быть усечена). Отметим, что если пароль будет пустой строкой, то таким же будет и P.

4. Устанавливаем, что $I = S||P$ соединением S и P.

5. Устанавливаем, что $j = \lceil n/u \rceil$.

6. Для $i = 1, 2, \ldots, j$, делаем следующее:

a) устанавливаем, что $A_i = \mathrm{H}^c(D||I)$, c-й хэш $D||I$. То есть вычисляем хэш $D||I$; вычисляем хэш этого хэша и т. д. Продолжаем вычислять так же, пока не будет вычислено общее количество хэшей c,
каждый из которых является результатом предыдущего;

b) соединяем копии A_i для создания строки B длиной в v бит (для создания B замыкающая копия A_i может быть усечена);

c) рассматривая I как связку $I_0, I_1, \ldots, I_{k-1}$ v-битных блоков, где $k = \lceil s/v \rceil + \lceil p/v \rceil$, изменяем I, устанавливая, что $I_j = (I_j + B + 1) \bmod 2^v$ для каждого j. Чтобы выполнить это сложение, рассматриваем каждый v-битный блок как число в двоичной форме представления, где наиболее старший по разряду бит стоит первым.

7. Соединяем вместе $A_1, A_2, \ldots, A_j$ для формирования псевдослучайной битовой строки A.

8. Используем первые n бит из A как выводную последовательность всего этого процесса.

Когда представленные в настоящем разделе механизмы шифрования, основанные на пароле, используются для генерации ключа и вектора инициализации IV (если необходимо) из пароля, величины «salt» и отсчета итераций, используют вышеупомянутый алгоритм. Для генерации ключа байт идентификатора ID устанавливается на значение 1; для генерации вектора инициализации IV байт идентификатора ID устанавливается на значение 2.

Когда представленный в настоящем разделе механизм аутентификации используется для гене-

рации ключа из пароля, величины «salt» и отсчета итераций, используют вышеупомянутый алгоритм. Байт идентификатора *ID* устанавливается на значение 3.

12.27.1 SHA-1-PBE для 128-битного RC4

Механизм SHA-1-PBE для 128-битного RC4, обозначенный как **CKM_PBE_SHA1_RC4_128**, представляет собой механизм, используемый для генерации 128-битного секретного ключа RC4 из пароля и значения «salt» с использованием алгоритма списка SHA-1 и отсчета итераций. Метод, используемый для генерации ключа, описан выше.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей. Данный параметр имеет также поле для расположения используемого приложением буфера, в который поступит сгенерированный данным механизмом вектор инициализации IV; данным механизмом содержимое данного поля игнорируется, поскольку RC4 не требует вектора инициализации IV.

Ключ, генерируемый данным механизмом, будет, как правило, использоваться для осуществления шифрования на основе пароля.

12.27.2 SHA-1-PBE для 40-битного RC4

Механизм SHA-1-PBE для 40-битного RC4, обозначенный как **CKM_PBE_SHA1_RC4_40**, представляет собой механизм, используемый для генерации 40-битного секретного ключа RC4 из пароля и значения «salt» с использованием алгоритма списка SHA-1 и отсчета итераций. Метод, используемый для генерации ключа, описан выше.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей. Данный параметр имеет также поле для расположения используемого приложением буфера, в который поступит сгенерированный данным механизмом вектор инициализации IV; данным механизмом содержимое данного поля игнорируется, поскольку RC4 не требует вектор инициализации IV.

Ключ, генерируемый данным механизмом, будет, как правило, использоваться для осуществления шифрования на основе пароля.

12.27.3 SHA-1-PBE для 3-ключевого тройного DES-CBC

Механизм SHA-1-PBE для 3-ключевого тройного DES-CBC, обозначенный как **CKM_PBE_SHA1_DES3_EDE_CBC**, представляет собой механизм, используемый для генерации секретного ключа 3-ключе-вого тройного DES и вектора инициализации IV из пароля и значения «salt» с использованием алгоритма списка SHA-1 и отсчета итераций. Метод, используемый для генерации ключа и IV, описан выше. У каждого байта произведенного ключа будет, если необходимо, скорректирован младший бит, чтобы в результате был получен корректный ключ 3-ключевого тройного DES с верными битами четности.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

Ключ и вектор инициализации IV, генерируемые данным механизмом, будут, как правило, использоваться для осуществления шифрования на основе пароля.

12.27.4 SHA-1-PBE для 2-ключевого тройного DES-CBC

Механизм SHA-1-PBE для 2-ключевого тройного DES-CBC, обозначенный как **CKM_PBE_SHA1_DES2_EDE_CBC**, представляет собой механизм, используемый для генерации секретного ключа 2-ключе-вого тройного DES и вектора инициализации IV из пароля и значения «salt» с использованием алгоритма списка SHA-1 и отсчета итераций. Метод, используемый для генерации ключа и вектора инициализации IV, описан выше. У каждого байта произведенного ключа будет, если необходимо, скорректирован младший бит, чтобы в результате был получен корректный ключ 2-ключевого тройного DES с верными битами четности.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

Ключ и вектор инициализации IV, генерируемые данным механизмом, будут, как правило, использоваться для осуществления шифрования на основе пароля.

12.27.5 SHA-1-PBE для 128-битного RC2-CBC

Механизм SHA-1-PBE для 128-битного RC2-CBC, обозначенный как **CKM_PBE_SHA1_RC2_128_CBC**, представляет собой механизм, используемый для генерации 128-битного секретного ключа RC2 и

вектора инициализации IV из пароля и значения «salt» с использованием алгоритма списка SHA-1 и отсчета итераций. Метод, используемый для генерации ключа и вектора инициализации IV, описан выше.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

Когда ключ и вектор инициализации IV, сгенерированные данным механизмом, используются для шифрования или расшифрования, эффективное количество бит в поисковом пространстве RC2 должно быть установлено на 128. Это обеспечит совместимость с идентификатором объекта ASN.1 `pbeWithSHA1And128BitRC2-CBC`.

Ключ и IV, генерируемые данным механизмом, будут, как правило, использоваться для осуществления шифрования на основе пароля.

12.27.6 SHA-1-PBE для 40-битного RC2-CBC

Механизм SHA-1-PBE для 40-битного RC2-CBC, обозначенный как **CKM_PBE_SHA1_RC2_40_CBC**, представляет собой механизм, используемый для генерации 40-битного секретного ключа RC2 и вектора инициализации IV из пароля и значения «salt» с использованием алгоритма списка SHA-1 и отсчета итераций. Метод, используемый для генерации ключа и вектора инициализации IV, описан выше.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей и расположение используемого приложением буфера, в который поступит сгенерированный данным механизмом 8-байтный вектор инициализации IV.

Когда ключ и вектор инициализации IV, сгенерированные данным механизмом, используются для шифрования или расшифрования, эффективное количество бит в поисковом пространстве RC2 должно быть установлено на 40. Это обеспечит совместимость с идентификатором объекта ASN.1 `pbeWithSHA1And128BitRC2-CBC`.

Ключ и вектор инициализации IV, генерируемые данным механизмом, будут, как правило, использоваться для осуществления шифрования на основе пароля.

12.27.7 SHA-1-PBA для SHA-1-HMAC

Механизм SHA-1-PBA для SHA-1-HMAC, обозначенный как **CKM_PBA_SHA1_WITH_SHA1_HMAC**, представляет собой механизм, используемый для генерации 160-битного общего секретного ключа из пароля и значения «salt» с использованием алгоритма списка SHA-1 и отсчета итераций. Метод, используемый для генерации ключа, описан выше.

Данный механизм имеет параметр структуры **CK_PBE_PARAMS**. Данный параметр задает информацию ввода для процесса генерации ключей. Данный параметр имеет также поле для расположения используемого приложением буфера, в который поступит сгенерированный данным механизмом вектор инициализации IV; данным механизмом содержимое данного поля игнорируется, поскольку аутентификация с помощью SHA-1-HMAC не требует вектора инициализации IV.

Ключ, генерируемый данным механизмом, будет, как правило, использоваться для вычисления SHA-1 HMAC с целью осуществления аутентификации на основе пароля (а не *шифрования на основе пароля*). На момент написания данного документа это, как правило, делается для обеспечения целостности PKCS #12 PDU.

12.28 RIPE-MD

12.28.1 Определения

Механизмы:

```
CKM_RIPEMD128
CKM_RIPEMD128_HMAC
CKM_RIPEMD128_HMAC_GENERAL
CKM_RIPEMD160
CKM_RIPEMD160_HMAC
CKM_RIPEMD160_HMAC_GENERAL
```

12.28.2 Список RIPE-MD 128

Механизм RIPE-MD 128, именуемый **CKM_RIPEMD128**, является механизмом получения списка сообщения посредством применения алгоритма получения списка сообщения RIPE-MD 128.

Данный механизм не имеет параметра.

Ограничения на длину данных приведены в таблице 139.

Таблица 139 – RIPE-MD 128: Длина данных

Функция	Длина данных	Длина списка
C_Digest	Любая	16

12.28.3 Механизм общей длины RIPE-MD 128-HMAC

Механизм общей длины RIPE-MD 128-HMAC, обозначенный как **CKM_RIPEMD128_HMAC_GENERAL**, представляет собой механизм, используемый для постановки подписей и осуществления верификации. Данный механизм использует конструкцию HMAC, основанную на функции хэширования RIPE-MD 128. Используемые данным механизмом ключи являются общими секретными ключами.

Данный механизм имеет параметр **CK_MAC_GENERAL_PARAMS**, который содержит длину желаемого выводного массива в байтах. Эта длина должна быть в диапазоне 0 – 16 (размер выводного массива RIPE-MD 128 составляет 16 байт). Подписи (MAC), производимые данным механизмом, будут браться из начальной части полного 16-байтного выводного массива HMAC.

Таблица 140 – Механизм общей длины RIPE-MD 128-HMAC

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	Общий секретный	Любая	0 – 16, в зависимости от параметров
C_Verify	Общий секретный	Любая	0 – 16, в зависимости от параметров

12.28.4 RIPE-MD 128-HMAC

Механизм RIPE-MD 128-HMAC, обозначенный как **CKM_RIPEMD128_HMAC**, представляет собой частный случай механизма общей длины RIPE-MD 128-HMAC, описанного в 12.28.3.

Данный механизм не имеет параметра и всегда производит выводной массив длиной 16 байт.

12.28.5 RIPE-MD 160

Механизм RIPE-MD 128-HMAC, обозначенный как **CKM_RIPEMD128_HMAC**, представляет собой частный случай механизма общей длины RIPE-MD 128-HMAC, описанного в 12.28.3.

Данный механизм не имеет параметра

Механизм RIPE-MD 160, обозначенный как **CKM_RIPEMD160**, представляет собой механизм для получения списка сообщений при помощи алгоритма списка RIPE-MD 160, описанного в ISO 10118.

Данный механизм не имеет параметра.

Ограничения на длину данных приведены в таблице 141.

Таблица 141 – RIPE-MD 160: Длина данных

Функция	Длина данных	Длина списка
C_Digest	Любая	20

12.28.6 Механизм общей длины RIPE-MD 160-HMAC

Механизм общей длины RIPE-MD 160-HMAC, обозначенный как **CKM_RIPEMD160_HMAC_GENERAL**, представляет собой механизм, используемый для постановки подписей и осуществления верификации. Данный механизм использует конструкцию HMAC, основанную на функции хэширования RIPE-MD 160. Используемые данным механизмом ключи являются общими секретными ключами.

Данный механизм имеет параметр **CK_MAC_GENERAL_PARAMS**, который содержит длину желаемого выводного массива в байтах. Эта длина должна быть в диапазоне 0 – 20 (размер выводного массива RIPE-MD 160 составляет 20 байт). Подписи (MAC), производимые данным механизмом, будут браться из начальной части полного 20-байтного выводного массива HMAC.

Таблица 142 – Механизм общей длины RIPE-MD 160-HMAC

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	Общий секретный	Любая	0 – 20, в зависимости от параметров
C_Verify	Общий секретный	Любая	0 – 20, в зависимости от параметров

12.28.7 RIPE-MD 160-HMAC

Механизм RIPE-MD 160-HMAC, обозначенный как **CKM_RIPEMD160_HMAC**, представляет собой частный случай механизма общей длины RIPE-MD 160-HMAC, описанного в 12.28.6.

Данный механизм не имеет параметра и всегда производит выводной массив длиной 20.

12.29 SET

12.29.1 Определения

Механизмы:

```
CKM_KEY_WRAP_SET_OAEP
```

12.29.2 Параметры механизма SET

♦ **CK_KEY_WRAP_SET_OAEP_PARAMS; CK_KEY_WRAP_SET_OAEP_PARAMS_PTR**

CK_KEY_WRAP_SET_OAEP_PARAMS является структурой, предоставляющей параметры механизму **CKM_KEY_WRAP_SET_OAEP**. Она задается следующим образом:

```
typedef struct CK_KEY_WRAP_SET_OAEP_PARAMS {
  CK_BYTE bBC;
  CK_BYTE_PTR pX;
  CK_ULONG ulXLen;
} CK_KEY_WRAP_SET_OAEP_PARAMS;
```

Поля данной структуры имеют следующие значения:

bBC байт содержания блока;

pX последовательность хэш-значений данных открытого текста (если имеются) и дополнительных данных (если имеются);

ulXLen длина в байтах последовательности хэш-значений данных открытого текста (если имеются) и дополнительных данных (если имеются). Если ни одного из значений не имеется, значение поля – 0.

CK_KEY_WRAP_SET_OAEP_PARAMS_PTR являетя указателем на **CK_KEY_WRAP_SET_OAEP_PARAMS**.

12.29.3 Упаковка ключа OAEP для SET

Механизм упаковки ключа OAEP для SET, обозначенный как **CKM_KEY_WRAP_SET_OAEP**, представляет собой механизм для упаковки и распаковки ключа DES ключом RSA. Вместе с данным ключом DES могут также в необязательном порядке упаковываться хэш-значения некоторых данных открытого текста и/или дополнительных данных. Данный механизм определен в описаниях протокола SET.

Данный механизм принимает параметр структуры **CK_KEY_WRAP_SET_OAEP_PARAMS**. Данная структура содержит байт «Содержание блока» с данными и последовательностью хэш-значений данных открытого текста (если имеются) и дополнительных данных, подлежащих упаковке (если имеются). Если не имеется ни хэш-значений, ни дополнительных данных, это отображается полем *ulXLen*, принимающим значение 0.

Когда данный механизм используется для распаковки ключа, последовательность хэш-значений открытого текста (если имеются) и дополнительных данных (если имеются) выдается по соглашению, описанному в 11.2, о формировании выводного массива данных. Следует отметить, что данные, вводимые в **C_UnwrapKey**, таковы, что дополнительные данные не выдаются (*например*, буфер, предоставленный в структуре **CK_KEY_WRAP_SET_OAEP_PARAMS,** имеет содержимое NULL_PTR) и объект распакованного ключа, в свою очередь, также создан не будет.

Имейте в виду, что, когда данный механизм используется для распаковки ключа, поля *bBC* и *pX* параметра, подаваемого в данный механизм, могут быть модифицированы.

Если приложение применяет **C_UnwrapKey** с **CKM_KEY_WRAP_SET_OAEP**, для приложения может быть предпочтительнее просто предоставить 128-байтный буфер для последовательности хэш-значений данных открытого текста и дополнительных данных (эта последовательность никогда не бывает больше 128 байт), чем вызывать **C_UnwrapKey** дважды. Каждый вызов **C_UnwrapKey** с **CKM_KEY_WRAP_SET_OAEP** требует выполнения операции расшифрования RSA, а описанным способом можно избежать этих чрезмерных вычислений.

12.30 LYNKS

12.30.1 Определения

Механизмы:

```
CKM_KEY_WRAP_LYNKS
```

12.30.2 Упаковка ключа LYNKS

Механизм упаковки ключей LYNKS, обозначенный как **CKM_KEY_WRAP_LYNKS**, представляет собой механизм для упаковки и распаковки секретных ключей при помощи ключей DES. Данный механизм может упаковать любой 8-байтный секретный ключ, производя при этом 10-байтный упакованный ключ, содержащий криптографическую контрольную сумму.

Данный механизм не имеет параметра.

Чтобы упаковать 8-байтный секретный ключ *K* ключом DES *W*, данный механизм осуществляет следующие шаги:

1. Производится инициализация (задаются начальные значения) двух 16-битных целых величин sum_1 и sum_2 до 0.
2. Выполняется цикл через байты ключа *K* от первого до последнего.
3. Устанавливается, что $sum_1 = sum_1$ + байт ключа (рассматривая этот байт ключа как число диапазона 0 – 255).
4. Устанавливается, что $sum_2 = sum_2 + sum_1$.
5. *K* шифруется ключом *W* в режиме ECB, получается зашифрованный ключ *E*.
6. Последние 6 байт *E* объединяются с sum_2, при этом наиболее старший по разряду бит sum_2 идет первым. Результатом будет 8-байтный блок *T*.
7. *T* шифруется ключом *W* в режиме ECB, получаем зашифрованную контрольную сумму *C*.
8. *E* объединяется с последними 2 байтами *C* для получения упакованного ключа.

Если при распаковке ключа при помощи данного механизма криптографическая контрольная сумма вычисляется неверно, выдается сообщение об ошибке. Кроме того, если при помощи данного механизма распаковывается ключ DES или ключ CDMF, должны быть верно установлены биты четности упакованного ключа. Если они установлены неверно, выдается сообщение об ошибке.

12.31 SSL

12.31.1 Определения

Механизмы:

```
CKM_SSL3_PRE_MASTER_KEY_GEN
CKM_SSL3_MASTER_KEY_DERIVE
CKM_SSL3_KEY_AND_MAC_DERIVE
CKM_SSL3_MASTER_KEY_DERIVE_DH
CKM_SSL3_MD5_MAC
CKM_SSL3_SHA1_MAC
```

12.31.2 Параметры механизма SSL

♦ **CK_SSL3_RANDOM_DATA**

CK_SSL3_RANDOM_DATA является структурой, которая предоставляет информацию о случайных данных клиента и сервера в контекст SSL. Данная структура используется обоими механизмами: **CKM_SSL3_MASTER_KEY_DERIVE** и **CKM_SSL3_KEY_AND_MAC_DERIVE**. Данная структура задается следующим образом:

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

Поля данной структуры имеют следующие значения:

pClientRandom указатель на случайные данные клиента;
ulClientRandomLen длина в байтах случайных данных клиента;
pServerRandom указатель на случайные данные сервера;
ulServerRandomLen длина в байтах случайных данных сервера.

♦ **CK_SSL3_MASTER_KEY_DERIVE_PARAMS; CK_SSL3_MASTER_KEY_DERIVE_PARAMS_PTR**

CK_SSL3_MASTER_KEY_DERIVE_PARAMS является структурой, которая предоставляет параметры механизму **CKM_SSL3_MASTER_KEY_DERIVE**. Данная структура задается следующим образом:

```
typedef struct CK_SSL3_MASTER_KEY_DERIVE_PARAMS {
  CK_SSL3_RANDOM_DATA RandomInfo;
  CK_VERSION_PTR pVersion;
} CK_SSL3_MASTER_KEY_DERIVE_PARAMS;
```

Поля данной структуры имеют следующие значения:

RandomInfo информация случайных данных клиента и сервера;

pVersion указатель на структуру **CK_VERSION**, которая получает информацию о версии протокола SSL.

CK_SSL3_MASTER_KEY_DERIVE_PARAMS_PTR является указателем на **CK_SSL3_MASTER_KEY_DERIVE_PARAMS**.

♦ **CK_SSL3_KEY_MAT_OUT; CK_SSL3_KEY_MAT_OUT_PTR**

CK_SSL3_KEY_MAT_OUT является структурой, которая содержит результирующие идентификаторы ключей (key handles) и векторы инициализации после выполнения фунции C_DeriveKey с механизмом **CKM_SSL3_KEY_AND_MAC_DERIVE**. Данная структура определяется следующим образом:

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

Поля данной структуры имеют следующие значения:

hClientMacSecret идентификатор (key handle) для результирующего ключа MAC клиента;

hServerMacSecret идентификатор (key handle) для результирующего ключа MAC сервера;

hClientKey идентификатор (key handle) для результирующего секретного ключа клиента;

hServerKey идентификатор (key handle) для результирующего секретного ключа сервера;

pIVClient указатель на участок памяти, получающий вектор инициализации(I), созданный для клиента (если имеется);

pIVServer указатель на участок памяти, получающий вектор инициализации(I), созданный для сервера (если имеется).

CK_SSL3_KEY_MAT_OUT_PTR является указателем на **CK_SSL3_KEY_MAT_OUT**.

♦ **CK_SSL3_KEY_MAT_PARAMS; CK_SSL3_KEY_MAT_PARAMS_PTR**

CK_SSL3_KEY_MAT_PARAMS является структурой, предоставляющей параметры механизму **CKM_SSL3_KEY_AND_MAC_DERIVE**. Данная структура определяется следующим образом:

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

Поля данной структуры имеют следующие значения:

ulMacSizeInBits длина (в битах) ключей MAC, согласованных в ходе осуществления в соответствии с протоколом процедуры квитирования установления связи;

ulKeySizeInBits длина (в битах) секретных ключей, согласованных в ходе осуществления в соответствии с протоколом процедуры квитирования установления связи;

ulIVSizeInBits длина (в битах) IV, согласованных в ходе осуществления в соответствии с протоколом процедуры квитирования установления связи. Если IV не требуется, длина должна быть установлена в размере 0;

bIsExport булева величина, указывающая, должны ли ключи выводиться для

	экспортной версии протокола;
RandomInfo	информация о случайных данных клиента и сервера;
pReturnedKeyMaterial	указатели на структуры **CK_SSL3_KEY_MAT_OUT**, которые получают идентификаторы сгенерированных ключей и IV.

CK_SSL3_KEY_MAT_PARAMS_PTR является указателем на **CK_SSL3_KEY_MAT_PARAMS**.

12.31.3 Генерация ключа «pre_master»

Генерация ключа «pre_master» в SSL 3.0, обозначенная как **CKM_SSL3_PRE_MASTER_KEY_GEN**, представляет собой механизм, который генерирует 48-байтный общий секретный ключ. Данный механизм применяется для выработки ключа «pre_master», используемого в SSL версии 3.0 для шифровальных схем, подобных RSA.

Данный механизм имеет один параметр структуры **CK_VERSION**, который выдает номер версии SSL, используемой клиентом.

Данный механизм добавляет к новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут быть приданы значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_GenerateKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** имеет значение 48. Однако, поскольку все эти факты подразумеваются в механизме, нет необходимости указывать ни на один из них.

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 48 байт.

12.31.4 Деривация мастер-ключа

Деривация мастер-ключа в SSL 3.0, обозначенная как **CKM_SSL3_MASTER_KEY_DERIVE**, представляет собой механизм, применяемый для деривации одного 48-байтного общего секретного ключа из другого 48-байтного общего секретного ключа. Данный механизм используется для выработки ключа «master_secret», применяемого в протоколе SSL, из ключа «pre_master». Данный механизм выдает значение клиентской версии, которое встраивается в ключ «pre_master», а также идентификатор полученного ключа «master_secret».

Данный механизм имеет параметр структуры **CK_SSL3_MASTER_KEY_DERIVE_PARAMS**, который позволяет передавать случайные данные на носитель, а также выдавать номер версии протокола, являющийся частью ключа «pre_master». Данная структура описана в 12.31.

Данный механизм предоставляет новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут быть приданы значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_DeriveKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** имеет значение 48. Однако, поскольку все эти факты подразумеваются в механизме, нет необходимости указывать ни на один из них.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 48 байт.

Структура **CK_VERSION**, на которую указывает поле *pVersion* структуры **CK_SSL3_MASTER_KEY_DERIVE_PARAMS**, будет модифицирована запросом **C_DeriveKey**. В частности, когда выдается запрос, данная структура будет содержать версию SSL, ассоциированную с предоставленным ключом «pre_master».

Следует отметить, что данный механизм применим только к шифрсхемам, использующим 48-

байтный ключ «master_secret» со встроенным номером версии. Это относится к шифр-схемам RSA, однако исключается применение данного механизма с шифрсхемами Диффи – Хеллмана.

12.31.5 Деривация мастер-ключа для шифрсхем Диффи – Хеллмана

Деривация мастер-ключа для шифрсхем Диффи – Хеллмана в SSL 3.0, обозначенная как **CKM_SSL3_MASTER_KEY_DERIVE_DH**, представляет собой механизм, применяемый для деривации одного 48-байтного общего секретного ключа из другого общего секретного ключа произвольной длины. Данный механизм используется для выработки ключа «master_secret», применяемого в протоколе SSL, из ключа «pre_master».

Данный механизм имеет параметр структуры **CK_SSL3_MASTER_KEY_DERIVE_PARAMS**, который позволяет передавать случайные данные на носитель. Данная структура описана в 12.31. Для поля *pVersion* данной структуры должно быть задано значение NULL_PTR, поскольку номер версии не встроен в ключ «pre_master», как это сделано в шифрсхемах, подобных RSA.

Данный механизм предоставляет новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут быть приданы значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_DeriveKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** имеет значение 48. Однако, поскольку все эти факты подразумеваются в механизме, нет необходимости указывать ни на один из них.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 48 байт.

Следует отметить, что данный механизм применим только к шифрсхемам, которые не используют 48-байтный ключ «master_secret» со встроенным номером версии. Это относится к шифрсхемам Диффи – Хеллмана, однако исключается применение данного механизма с шифрсхемами RSA.

12.31.6 Деривация ключа и MAC

Деривация ключа, MAC и вектора инициализации IV в SSL 3.0, обозначенная как **CKM_SSL3_KEY_AND_MAC_DERIVE**, представляет собой механизм, используемый для выведения соответствующего криптографического ключевого материала, используемого структурой «CipherSuite» («Шифрсхема»), из данных «секретного ключа» и случайных данных. Данный механизм выдает идентификаторы для ключей, генерируемых в данном процессе, а также для создаваемых векторов инициализации IV.

Данный механизм имеет параметр структуры **CK_SSL3_KEY_MAT_PARAMS**, который позволяет передавать случайные данные, а также характеристики криптографического материала для заданной шифрсхемы, а также передавать указатель на структуру, которая получает сгенерированные идентификаторы и векторы инициализации. Данная структура описана в 12.31.

Данный механизм вносит вклад в создание на носителе четырех отдельных ключей и выдает автору запроса два вектора инициализации IV (если они запрашиваются автором запроса). Всем этим ключам придается класс объекта **CKO_SECRET_KEY**.

Двум ключам MAC («client_write_MAC_secret» и «server_write_MAC_secret») всегда придаются типы **CKK_GENERIC_SECRET**. Они обозначаются флагами как пригодные для операций электронной подписи, верификации и деривации.

Двум другим ключам («client_write_key» и «server_write_key») придаются типы в соответствии с информацией, обнаруженной в шаблоне, посылаемом вместе с данным механизмом в ходе выполнения функции запроса **C_DeriveKey**. По умолчанию они обозначаются флагами как пригодные для

операций шифрования, расшифрования и деривации.

Векторы инициализации (IV) будут генерироваться и выдаваться, если поле *ulIVSizeInBits* в структуре **CK_SSL_KEY_MAT_PARAMS** имеет не нулевое значение. Если векторы инициализации IV генерируются, их длина в битах будет согласовываться с величиной, содержащейся в поле *ulIVSizeInBits*.

Все четыре ключа наследуют у базового ключа значения его атрибутов **CKA_SENSITIVE**, **CKA_ALWAYS_SENSITIVE**, **CKA_EXTRACTABLE** и **CKA_NEVER_EXTRACTABLE**. Шаблон, предоставляемый для **C_DeriveKey**, может не определять значения для любого из этих атрибутов, которые отличаются от значений базового ключа.

Следует отметить, что структура **CK_SSL3_KEY_MAT_OUT**, на которую указывает поле *pReturnedKeyMaterial* структуры **CK_SSL3_KEY_MAT_PARAMS**, будет модифицирована запросом **C_DeriveKey**. В частности, четыре идентификатора ключей в структуре **CK_SSL3_KEY_MAT_OUT** будут модифицированы таким образом, чтобы данная структура содержала идентификаторы для вновь создаваемых ключей; в дополнение к этому, буферы, на которые указывают поля *pIVClient* и *pIVServer* структуры **CK_SSL3_KEY_MAT_OUT**, будут содержать выданные в них векторы инициализации IV (если они запрашивались автором запроса). Следовательно, эти два поля должны указывать на буферы, имеющие достаточно места для размещения любого вектора инициализации IV, который будет выдан.

Данный механизм дистанцируется от остальных механизмов деривации ключей в Cryptoki по выдаваемой им информации. Для большинства механизмов деривации ключа **C_DeriveKey** выдает в качестве успешного результата завершения операции один идентификатор ключа. Однако, поскольку механизм **CKM_SSL3_KEY_AND_MAC_DERIVE** выдает все свои идентификаторы ключа в структуру **CK_SSL3_KEY_MAT_OUT**, на которую указывает структура **CK_SSL3_KEY_MAT_PARAMS**, заданная как параметр механизма, параметр *phKey*, передаваемый в **C_DeriveKey,** не нужен, и его содержимое должно быть NULL_PTR.

Если запрос данного механизма к **C_DeriveKey** не удовлетворяется, тогда на носителе не будет создан *ни один* из четырех ключей.

12.31.7 Механизм MD5 MAC в SSL 3.0

Механизм MD5 MAC в SSL3.0, обозначенный как **CKM_SSL3_MD5_MAC**, представляет собой механизм для односторонних и многосторонних процедур постановки электронной подписи (аутентификации данных) и верификации с использованием MD5 на основе протокола SSL 3.0. Эта методика очень похожа на методику HMAC.

Данный механизм имеет параметр **CK_MAC_GENERAL_PARAMS**, который определяет длину в байтах подписей, производимых данным механизмом.

Ограничения на типы ключа и длину данных ввода/вывода приведены в таблице 143.

Таблица 143 – MD5 MAC в SSL 3.0: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	Общий секретный	Любая	4 – 8, в зависимости от параметров
C_Verify	Общий секретный	Любая	4 – 8, в зависимости от параметров

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** указывают поддерживаемый диапазон размеров общих секретных ключей в битах.

12.31.8 Механизм SHA-1 MAC в SSL 3.0

Механизм SHA-1 MAC в SSL3.0, обозначенный как **CKM_SSL3_SHA1_MAC**, представляет собой механизм для односторонних и многосторонних процедур постановки электронной подписи (аутентификации данных) и верификации с использованием SHA-1 на основе протокола SSL 3.0. Эта методика очень похожа на методику HMAC.

Данный механизм имеет параметр **CK_MAC_GENERAL_PARAMS**, который определяет длину в байтах подписей, производимых данным механизмом.

Ограничения на типы ключа и длину данных ввода/вывода приведены в таблице 144.

Таблица 144 – SHA-1 MAC в SSL 3.0: Длина ключа и данных

Функция	Тип ключа	Длина данных	Длина подписи
C_Sign	Общий секретный	Любая	4 – 8, в зависимости от параметров
C_Verify	Общий секретный	Любая	4 – 8, в зависимости от параметров

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO**

указывают поддерживаемый диапазон размеров общих секретных ключей в битах.

12.32 TLS

Подробности в [TLS].

12.32.1 Определения

Механизмы:
```
CKM_TLS_PRE_MASTER_KEY_GEN
CKM_TLS_MASTER_KEY_DERIVE
CKM_TLS_KEY_AND_MAC_DERIVE
CKM_TLS_MASTER_KEY_DERIVE_DH
CKM_TLS_PRF
```

12.32.2 Параметры механизма TLS

♦ **CK_TLS_PRF_PARAMS; CK_TLS_PRF_PARAMS_PTR**

CK_TLS_PRF_PARAMS является структурой, которая предоставляет параметры механизму **CKM_TLS_PRF**. Данная структура задается следующим образом:

```
typedef struct CK_TLS_PRF_PARAMS {
  CK_BYTE_PTR pSeed;
  CK_ULONG ulSeedLen;
  CK_BYTE_PTR pLabel;
  CK_ULONG ulLabelLen;
  CK_BYTE_PTR pOutput;
  CK_ULONG_PTR pulOutputLen;
} CK_TLS_PRF_PARAMS;
```

Поля данной структуры имеют следующие значения:

pSeed	указатель на вводимое начальное число (seed);
ulSeedLen	длина начального числа в байтах;
PLabel	указатель на идентифицирующую метку;
ulLabelLen	длина идентифицирующей метки в байтах;
POutput	указатель, получающий выходной результат операции;
pulOutputLen	указатель на длину в байтах, которую должен иметь создаваемый выводной массив, данный указатель должен иметь желаемую длину, как и ввод, и ему будет придаваться вычисленная длина как выводу.

CK_TLS_PRF_PARAMS_PTR является указателем на **CK_TLS_PRF_PARAMS**.

12.32.3 TLS PRF (псевдослучайная функция)

Псевдослучайная функция (pseudo random function, PRF) в TLS, обозначенная как **CKM_TLS_PRF**, является механизмом, используемым для создания генерируемых в защищенном режиме превдослучайных выводных массивов произвольной длины. Ключи, используемые данным механизмом, являются общими секретными ключами.

Данный механизм имеет параметр структуры **CK_TLS_PRF_PARAMS**, позволяющий передачу вводимого начального числа (seed) и его длины, передачу идентификационной метки и ее длины, а также передачу длины выводного массива на носитель и получение выводного массива.

Данный механизм производит генерируемый в защищенном режиме псевдослучайный выводной массив, длина которого определена в соответствующем параметре.

Данный механизм дистанцируется от остальных механизмов деривации ключей в Cryptoki, так как он не использует отправку шаблона одновременно с запросом функции **C_DeriveKey**, что означает отправку шаблона с содержимым NULL_PTR. Для большинства механизмов деривации ключа **C_DeriveKey** выдает в качестве успешного результата завершения операции один идентификатор ключа. Однако, поскольку механизм **CKM_TLS_PRF** выдает запрашиваемое количество байт выводного массива во все свои идентификаторы ключа в структуру **CK_TLS_PRF_PARAMS**, заданную как параметр механизма, параметр *phKey*, передаваемый в **C_DeriveKey,** не нужен и его содержимое должно быть NULL_PTR.

Если запрос данного механизма к **C_DeriveKey** не удовлетворяется, тогда выводной массив сгенерирован не будет.

12.32.4 Генерация ключа «pre_master»

Генерация ключа «pre_master» в TLS 1.0, обозначенная как **CKM_SSL3_PRE_MASTER_KEY_GEN**,

представляет собой механизм, который генерирует 48-байтный общий секретный ключ. Данный механизм применяется для выработки ключа «pre_master», используемого в TLS version 1.0 для шифровальных схем, подобных RSA.

Данный механизм имеет один параметр структуры **CK_VERSION**, который выдает номер версии SSL, используемой клиентом.

Данный механизм добавляет к новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут быть приданы значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_GenerateKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** имеет значение 48. Однако, поскольку все эти факты подразумеваются в механизме, нет необходимости указывать ни на один из них.

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 48 байт.

12.32.5 Деривация мастер-ключа

Деривация мастер-ключа в TLS 1.0, обозначенная как **CKM_TLS_MASTER_KEY_DERIVE**, представляет собой механизм, применяемый для деривации одного 48-байтного общего секретного ключа из другого 48-байтного общего секретного ключа. Данный механизм используется для выработки ключа «master_secret», применяемого в протоколе TLS, из ключа «pre_master». Данный механизм выдает значение клиентской версии, которое встраивается в ключ «pre_master», а также идентификатор полученного ключа «master_secret».

Данный механизм имеет параметр структуры **CK_SSL3_MASTER_KEY_DERIVE_PARAMS**, который позволяет передавать случайные данные на носитель, а также выдавать номер версии протокола, являющийся частью ключа «pre_master». Данная структура описана в 12.31.

Данный механизм предоставляет новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут придаваться значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_GenerateKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** имеет значение 48. Однако, поскольку все эти факты подразумеваются в механизме, нет необходимости указывать ни на один из них.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 48 байт.

Следует отметить, что структура **CK_VERSION**, на которую указывает поле *pVersion* структуры **CK_SSL3_MASTER_KEY_DERIVE_PARAMS**, будет модифицирована запросом **C_DeriveKey**. В частности, когда выдается запрос, данная структура будет содержать версию SSL, ассоциированную с предоставленным ключом «pre_master».

Следует отметить, что данный механизм применим только к шифрсхемам, использующим 48-байтный секретный ключ «pre_master» со встроенным номером версии. Это относится к шифрсхемам RSA, однако исключается применение данного механизма с шифрсхемами Диффи – Хеллмана.

12.32.6 Деривация мастер-ключа для шифрсхем Диффи – Хеллмана

Деривация мастер-ключа для шифрсхем Диффи – Хеллмана в TLS 1.0SSL 3.0, обозначенная как **CKM_TLS_MASTER_KEY_DERIVE_DH**, представляет собой механизм, применяемый для деривации одного 48-байтного общего секретного ключа из другого общего секретного ключа произвольной

длины. Данный механизм используется для выработки ключа «master_secret», применяемого в протоколе TLS, из ключа «pre_master».

Данный механизм имеет параметр структуры **CK_SSL3_MASTER_KEY_DERIVE_PARAMS**, который позволяет передавать случайные данные на носитель. Данная структура описана в 12.31. Для поля *pVersion* данной структуры должно быть задано значение NULL_PTR, поскольку номер версии не встроен в ключ «pre_master», как это сделано в шифрсхемах, подобных RSA.

Данный механизм предоставляет новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут придаваться значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_DeriveKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** имеет значение 48. Однако, поскольку все эти факты подразумеваются в механизме, нет необходимости указывать ни на один из них.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.
- Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.
- Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 48 байт.

Следует отметить, что данный механизм применим только к шифрсхемам, которые не используют 48-байтный ключ «master_secret» со встроенным номером версии. Это относится к шифр-схемам Диффи – Хеллмана, однако исключается применение данного механизма с шифрсхемами RSA.

12.32.7 Деривация ключа и MAC

Деривация ключа, MAC и вектора инициализации IV в TLS 1.0, обозначенная как **CKM_TLS_KEY_ AND_MAC_DERIVE**, представляет собой механизм, используемый для выведения соответствующего криптографического ключевого материала, используемого структурой «CipherSuite» («Шифрсхема»), из данных «секретного ключа» и случайных данных. Данный механизм выдает идентификаторы для ключей, генерируемых в данном процессе, а также создаваемые векторы инициализации IV.

Данный механизм имеет параметр структуры **CK_SSL3_KEY_MAT_PARAMS**, который позволяет передавать случайные данные, а также характеристики криптографического материала для заданной шифрсхемы, а также передавать указатель на структуру, которая получает сгенерированные идентификаторы и векторы инициализации. Данная структура описана в 12.31.

Данный механизм вносит вклад в создание на носителе четырех отдельных ключей и выдает автору запроса два вектора инициализации IV (если они запрашиваются автором запроса). Всем этим ключам придается класс объекта **CKO_SECRET_KEY**.

Двум ключам MAC («client_write_MAC_secret» и «server_write_MAC_secret») всегда придаются типы **CKK_GENERIC_SECRET**. Они обозначаются флагами как пригодные для операций электронной подписи, верификации и деривации.

Двум другим ключам («client_write_key» и «server_write_key») придаются типы в соответствии с информацией, обнаруженной в шаблоне, посылаемом вместе с данным механизмом в ходе выполнения функции запроса **C_DeriveKey**. По умолчанию они обозначаются флагами как пригодные для операций шифрования, расшифровки и деривации.

Векторы инициализации IV будут генерироваться и выдаваться, если поле *ulIVSizeInBits* в структуре **CK_SSL_KEY_MAT_PARAMS** имеет ненулевое значение. Если векторы инициализации IV генерируются, их длина в битах будет согласовываться с величиной, содержащейся в поле *ulIVSizeInBits*.

Все четыре ключа наследуют у базового ключа значения его атрибутов **CKA_SENSITIVE**, **CKA_ALWAYS_SENSITIVE**, **CKA_EXTRACTABLE** и **CKA_NEVER_EXTRACTABLE**. Шаблон, предоставляемый для **C_DeriveKey**, может не определять значения для любого из этих атрибутов, которые

отличаются от значений базового ключа.

Следует отметить, что структура **CK_SSL3_KEY_MAT_OUT**, на которую указывает поле *pReturnedKeyMaterial* структуры **CK_SSL3_KEY_MAT_PARAMS**, будет модифицирована запросом **C_DeriveKey**. В частности, четыре идентификатора ключей в структуре **CK_SSL3_KEY_MAT_OUT** будут модифицированы таким образом, чтобы данная структура содержала идентификаторы для вновь создаваемых ключей; в дополнение к этому буферы, на которые указывают поля *pIVClient* и *pIVServer* структуры **CK_SSL3_KEY_MAT_OUT**, будут содержать выданные в них векторы инициализации IV (если они запрашивались автором запроса). Следовательно, эти два поля должны указывать на буферы, имеющие достаточно места для размещения любого вектора инициализации IV, который будет выдан.

Данный механизм дистанцируется от остальных механизмов деривации ключей в Cryptoki по выдаваемой им информации. Для большинства механизмов деривации ключа **C_DeriveKey** выдает в качестве успешного результата завершения операции один идентификатор ключа. Однако, поскольку механизм **CKM_SSL3_KEY_AND_MAC_DERIVE** выдает все свои идентификаторы ключа в структуру **CK_SSL3_KEY_MAT_OUT**, на которую указывает структура **CK_SSL3_KEY_MAT_PARAMS**, заданная как параметр механизма, параметр *phKey*, передаваемый в **C_DeriveKey,** не нужен и его содержимое должно быть NULL_PTR.

Если запрос данного механизма к **C_DeriveKey** не удовлетворяется, тогда на носителе не будет создан *ни один* из четырех ключей.

12.33 WTLS

Подробности в [WTLS].

При сравнении существующих механизмов TLS с данными расширениями, предназначенными для поддержки WTLS, кто-то может возразить, что нет необходимости иметь различимую идентификацию клиентской и серверной стороны процедуры квитирования связи. Однако, поскольку в WTLS сервер и клиент используют различные числа последовательностей, возможны случаи (например, когда WTLS используется для защиты асинхронных протоколов), где числа последовательностей на стороне клиента и на стороне сервера различаются, и, следовательно, это мотивирует предлагаемое разделение.

12.33.1 Определения

Механизмы:

```
CKM_WTLS_PRE_MASTER_KEY_GEN
CKM_WTLS_MASTER_KEY_DERIVE
CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC
CKM_WTLS_PRF
CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE
CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE
```

12.33.2 Параметры механизма WTLS

♦ **CK_WTLS_RANDOM_DATA; CK_WTLS_RANDOM_DATA_PTR**

CK_WTLS_RANDOM_DATA является структурой, предоставляющей информацию о случайных данных клиента и сервера в контексте WTLS. Данная структура используется механизмом **CKM_WTLS_MASTER_KEY_DERIVE**. Данная структура задается следующим образом:

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

Поля данной структуры имеют следующие значения:

pClientRandom	указатель на случайные данные клиента;
ulClientRandomLen	длина в байтах случайных данных клиента;
pServerRandom	указатель на случайные данные сервера;
ulServerRandomLen	длина в байтах случайных данных сервера.

CK_WTLS_RANDOM_DATA_PTR является указателем на **CK_WTLS_RANDOM_DATA.**

♦ **CK_WTLS_MASTER_KEY_DERIVE_PARAMS; CK_WTLS_MASTER_KEY_DERIVE_PARAMS _PTR**

CK_WTLS_MASTER_KEY_DERIVE_PARAMS является структурой, предоставляющей параметры механизму **CKM_WTLS_MASTER_KEY_DERIVE**. Она определяется следующим образом:

```
typedef struct CK_WTLS_MASTER_KEY_DERIVE_PARAMS {
  CK_MECHANISM_TYPE DigestMechanism;
  CK_WTLS_RANDOM_DATA RandomInfo;
  CK_BYTE_PTR pVersion;
} CK_WTLS_MASTER_KEY_DERIVE_PARAMS;
```

Поля данной структуры имеют следующие значения:

DigestMechanism тип механизма списка, который предстоит использовать (возможные типы можно найти в [WTLS]);

RandomInfo информация случайных данных клиента и сервера;

pVersion указатель на структуру **CK_BYTE**, которая получает информацию о версии протокола WTLS.

CK_WTLS_MASTER_KEY_DERIVE_PARAMS_PTR является указателем на **CK_WTLS_MASTER_KEY_DERIVE_PARAMS**.

♦ **CK_WTLS_PRF_PARAMS; CK_WTLS_PRF_PARAMS_PTR**

CK_WTLS_PRF_PARAMS является структурой, предоставляющей параметры механизму **CKM_WTLS_PRF**. Она определяется следующим образом:

```
typedef struct CK_WTLS_PRF_PARAMS {
  CK_MECHANISM_TYPE DigestMechanism;
  CK_BYTE_PTR pSeed;
  CK_ULONG ulSeedLen;
  CK_BYTE_PTR pLabel;
  CK_ULONG ulLabelLen;
  CK_BYTE_PTR pOutput;
  CK_ULONG_PTR pulOutputLen;
} CK_WTLS_PRF_PARAMS;
```

Поля данной структуры имеют следующие значения:

DigestMechanism тип механизма списка, который предстоит использовать (возможные типы можно найти в [WTLS]);

pSeed указатель на вводимое начальное число (seed);

ulSeedLen длина начального числа в байтах;

pLabel указатель на идентифицирующую метку;

ulLabelLen длина идентифицирующей метки в байтах;

pOutput указатель, получающий выходной результат операции;

pulOutputLen указатель на длину в байтах, которую должен иметь создаваемый выводной массив. Данный указатель должен иметь желаемую длину, как и ввод, и ему будет придаваться вычисленная длина как выводу.

CK_WTLS_PRF_PARAMS_PTR является указателем на **CK_WTLS_PRF_PARAMS**.

♦ **CK_WTLS_KEY_MAT_OUT; CK_WTLS_KEY_MAT_OUT_PTR**

CK_WTLS_KEY_MAT_OUT является структурой, которая содержит результирующие идентификаторы ключей и векторы инициализации после выполнения функции C_DeriveKey с механизмом **CKM_WTLS_SEVER_KEY_AND_MAC_DERIVE** или **CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE.** Данная структура задается следующим образом:

```
typedef struct CK_WTLS_KEY_MAT_OUT {
  CK_OBJECT_HANDLE hMacSecret;
  CK_OBJECT_HANDLE hKey;
  CK_BYTE_PTR pIV;
} CK_WTLS_KEY_MAT_OUT;
```

Поля данной структуры имеют следующие значения:

hMacSecret идентификатор результирующего секретного ключа MAC;

hKey идентификатор результирующего секретного ключа;

pIV указатель на участок памяти, который получает созданный вектор инициализации IV (если имеется).

CK_WTLS_KEY_MAT_OUT _PTR является указателем на **CK_WTLS_KEY_MAT_OUT**.

♦ **CK_WTLS_KEY_MAT_PARAMS; CK_WTLS_KEY_MAT_PARAMS_PTR**

CK_WTLS_KEY_MAT_PARAMS является структурой, предоставляющей параметры для механизмов

CKM_WTLS_SEVER_KEY_AND_MAC_DERIVE и **CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE**. Данная структура задается следующим образом:

```
typedef struct CK_WTLS_KEY_MAT_PARAMS {
  CK_MECHANISM_TYPE DigestMechanism;
  CK_ULONG ulMacSizeInBits;
  CK_ULONG ulKeySizeInBits;
  CK_ULONG ulIVSizeInBits;
  CK_ULONG ulSequenceNumber;
  CK_BBOOL bIsExport;
  CK_WTLS_RANDOM_DATA RandomInfo;
  CK_WTLS_KEY_MAT_OUT_PTR pReturnedKeyMaterial;
} CK_WTLS_KEY_MAT_PARAMS;
```

Поля данной структуры имеют следующие значения:

DigestMechanism тип механизма списка, который надлежит использовать (возможные типы механизма можно найти в [WTLS]);

ulMacSizeInBits длина (в битах) ключей MAC, согласованных в ходе осуществления в соответствии с протоколом процедуры квитирования установления связи;

ulKeySizeInBits длина (в битах) секретных ключей, согласованных в ходе осуществления в соответствии с протоколом процедуры квитирования установления связи;

ulIVSizeInBits длина (в битах) векторов инициализации IV, согласованных в ходе осуществления в соответствии с протоколом процедуры квитирования установления связи. Если вектор инициализации IV не требуется, длина должна быть установлена в размере 0;

ulSequenceNumber текущий номер последовательности, использованный для документа, направленого клиентом и сервером соответственно;

bIsExport булева величина, указывающая, должны ли ключи выводиться для экспортной версии протокола. Если эта величина является верной (т. е. ключи могут подлежать экспорту), тогда *ulKeySizeInBits* – длина ключа в битах перед расширением. Длина ключа после расширения определяется информацией, извлекаемой из шаблона, направляемого совместно с данным механизмом в ходе запроса функции C_DeriveKey (атрибут **CKA_KEY_TYPE** либо атрибут **CKA_VALUE_LEN**);

RandomInfo информация о случайных данных клиента и сервера;

pReturnedKeyMaterial указатели на структуру **CK_WTLS_KEY_MAT_OUT**, которая получает идентификаторы для генерируемых ключей и IV.

CK_WTLS_KEY_MAT_PARAMS_PTR является указателем на **CK_WTLS_KEY_MAT_PARAMS**.

12.33.3 Генерация ключа «pre_master» для схемы обмена ключами RSA

Генерация ключа «pre_master» для схемы обмена ключами RSA в WTLS, обозначенная как **CKM_WTLS_PRE_MASTER_KEY_GEN**, представляет собой механизм, который генерирует секретный ключ произвольной длины. Данный механизм применяется в процедуре выработки ключа «pre_master» для схемы обмена ключами RSA, используемой в WTLS. Данный механизм выдает идентификатор для ключа «master_secret».

Данный механизм имеет один параметр структуры **CK_VERSION**, который выдает клиентскую версию WTLS.

Данный механизм добавляет к новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут придаваться значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_GenerateKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** указывает длину ключа «pre_master».

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 48 байт.

12.33.4 Деривация секретного мастер – ключа

Деривация мастер-ключа в WTLS, обозначенная как **CKM_WTLS_MASTER_KEY_DERIVE**, представляет собой механизм, применяемый для деривации одного 20-байтного общего секретного ключа

из другого общего секретного ключа произвольной длины. Данный механизм используется для выработки ключа «master_secret», применяемого в WTLS, из ключа «master_secret». Данный механизм выдает значение клиентской версии, которое встраивается в ключ «pre_master», а также идентификатор полученного ключа «master_secret».

Данный механизм имеет параметр структуры **CK_WTLS_MASTER_KEY_DERIVE_PARAMS**, который позволяет передавать тип механизма списка, который предстоит использовать, а также передавать случайные данные на носитель и выдавать номер версии протокола, являющийся частью ключа «master_secret».

Данный механизм предоставляет новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут придаваться значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_DeriveKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** имеет значение 48. Однако, поскольку все эти факты подразумеваются в механизме, нет необходимости указывать ни на один из них.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.

Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.

Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 20 байт.

Следует отметить, что структура **CK_BYTE**, на которую указывает поле *pVersion* структуры **CK_WTLS_MASTER_KEY_DERIVE_PARAMS**, будет модифицирована запросом **C_DeriveKey**. В частности, когда выдается запрос, данная структура будет содержать версию WTLS, ассоциированную с предоставленным ключом «pre_master».

Следует отметить, что данный механизм применим только к шифрсхемам, использующим 20-байтный ключ «master_secret» со встроенным номером версии. Это относится к шифр-схемам RSA, однако исключается применение данного механизма со схемами обмена ключами Диффи – Хеллмана, а также со схемами обмена ключами, основанными на криптографии эллиптической кривой (Elliptic Curve Cryptography).

12.33.5 Деривация мастер-ключа для схем Диффи – Хеллмана и для схем, основанных на криптографии эллиптической кривой

Деривация ключа «master_secret» для схем Диффи – Хеллмана и схем, основанных на криптографии эллиптической кривой в WTLS, обозначенная как **CKM_WTLS_MASTER_KEY_DERIVE_DH_ECC**, представляет собой механизм, применяемый для деривации 20-байтного общего секретного ключа из секретного ключа произвольной длины. Данный механизм выдает идентификатор получен-ного ключа «master_secret».

Данный механизм имеет параметр структуры **CK_WTLS_MASTER_KEY_DERIVE_PARAMS**, который позволяет передавать на носитель тип механизма списка, который надлежит использовать, а также случайные данные. Для поля *pVersion* данной структуры должно быть задано значение NULL_PTR,
поскольку номер версии не встроен в ключ «master_secret», как это сделано в схемах обмена ключами, подобных RSA.

Данный механизм предоставляет новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE** (а также атрибут **CKA_VALUE_LEN**, если он не был предоставлен в шаблоне). Другие атрибуты могут быть определены в шаблоне, либо им могут придаваться значения по умолчанию.

Шаблон, посылаемый вместе с данным механизмом в ходе запроса **C_DeriveKey,** может указывать, что класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_GENERIC_SECRET**, а атрибут **CKA_VALUE_LEN** имеет значение 20. Однако, поскольку все эти факты подразумеваются в механизме, нет необходимости указывать ни на один из них.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

Оба атрибута **CKA_SENSITIVE** и **CKA_EXTRACTABLE** в шаблоне нового ключа могут быть установлены как CK_TRUE или CK_FALSE. Если этого нет, каждый из этих атрибутов примет значение по умолчанию.

Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_FALSE, тогда полученный ключ будет таким же. Если у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_ALWAYS_SENSITIVE** устанавливается на то же значение, что и его атрибут **CKA_SENSITIVE**.

Аналогично, если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_FALSE, тогда у полученного ключа будет происходить то же самое. Если у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** устанавливается как CK_TRUE, тогда у полученного ключа атрибут **CKA_EXTRACTABLE** примет *противоположное* значение **CKA_NEVER_EXTRACTABLE**.

Для данного механизма оба поля ulMinKeySize и ulMaxKeySize структуры **CK_MECHANISM_INFO** обозначают 20 байт.

Следует отметить, что данный механизм применим только к схемам, которые не используют для обмена ключами 20-байтный секретный ключ со встроенным номером версии. Это относится к схемам Диффи – Хеллмана и к схемам, основанным на криптографии эллиптической кривой, однако исключается применение данного механизма со схемами RSA.

12.33.6 WTLS PRF (псевдослучайная функция)

Псевдослучайная функция (pseudo random function, PRF) в WTLS, обозначенная как **CKM_WTLS_PRF**, является механизмом, используемым для создания генерируемых в защищенном режиме псевдослучайных выводных массивов произвольной длины. Ключи, используемые данным механизмом, являются общими секретными ключами.

Данный механизм имеет параметр структуры **CK_TLS_PRF_PARAMS**, позволяющий передачу на носитель типа механизма списка, который надлежит использовать, вводимого начального числа (seed) и его длины, передачу идентификационной метки и ее длины, а также передачу длины выводного

массива и получение выводного массива.

Данный механизм производит генерируемый в защищенном режиме псевдослучайный выводной массив, длина которого определена в соответствующем параметре.

Данный механизм дистанцируется от остальных механизмов деривации ключей в Cryptoki, так как он не использует отправку шаблона одновременно с запросом функции **C_DeriveKey**, что означает отправку шаблона с содержимым NULL_PTR. Для большинства механизмов деривации ключа **C_DeriveKey** выдает в качестве успешного результата завершения операции один идентификатор ключа. Однако, поскольку механизм **CKM_WTLS_PRF** выдает запрашиваемое количество байт выводного массива во все свои идентификаторы ключа в структуру **CK_WTLS_PRF_PARAMS**, заданную как параметр механизма, параметр *phKey*, передаваемый в **C_DeriveKey,** не нужен, и его содержимое должно быть NULL_PTR.

Если запрос данного механизма к **C_DeriveKey** не удовлетворяется, тогда выводной массив сгенерирован не будет.

12.33.7 Деривация серверного ключа и MAC

Деривация ключа, MAC и вектора инициализации IV в WTLS, обозначенная как **CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE**, представляет собой механизм, используемый для выведения соответствующего криптографического ключевого материала, используемого шифрсхемой, из данных секретного мастер – ключа и случайных данных. Данный механизм выдает идентификаторы для ключей, генерируемых в данном процессе, а также создаваемые векторы инициализации IV.

Данный механизм имеет параметр структуры **CK_WTLS_KEY_MAT_PARAMS**, который позволяет передавать тип механизма списка, который надлежит использовать, случайные данные, характеристики криптографического материала для заданной шифрсхемы, а также передавать указатель на структуру, которая получает сгенерированные идентификаторы и векторы инициализации.

Данный механизм вносит вклад в создание на носителе двух отдельных ключей и выдает автору запроса один вектор инициализации IV (если вектор инициализации IV запрашивается автором запроса). Всем этим ключам придается класс объекта **CKO_SECRET_KEY**.

Ключу MAC (server_write_MAC_secret) всегда придается тип **CKK_GENERIC_SECRET**. Он обозначается флагом как пригодный для операций электронной подписи, верификации и деривации.

Другому ключу (server_write_key) придается тип в соответствии с информацией, обнаруженной в шаблоне, посылаемом вместе с данным механизмом в ходе выполнения функции запроса **C_DeriveKey**. По умолчанию он обозначается флагом как пригодный для операций шифрования, расшифровки и деривации.

Вектор инициализации (server write IV) будет генерироваться и выдаваться, если поле *ulIVSizeInBits* в структуре **CK_WTLS_KEY_MAT_PARAMS** имеет ненулевое значение. Если вектор инициализации IV генерируется, его длина в битах будет согласовываться с величиной, содержащейся в поле *ulIVSizeInBits*.

Оба ключа наследуют у базового ключа значения его атрибутов **CKA_SENSITIVE**, **CKA_ALWAYS_ SENSITIVE**, **CKA_EXTRACTABLE** и **CKA_NEVER_EXTRACTABLE**. Шаблон, предоставляемый для **C_DeriveKey**, может не определять значения для любого из этих атрибутов, которые отличаются от значений базового ключа.

Следует отметить, что структура **CK_WTLS_KEY_MAT_OUT**, на которую указывает поле *pReturnedKeyMaterial* структуры **CK_WTLS_KEY_MAT_PARAMS**, будет модифицирована запросом **C_DeriveKey**. В частности, два поля идентификаторов ключей в структуре **CK_WTLS_KEY_MAT_OUT** будут модифицированы таким образом, чтобы данная структура содержала идентификаторы для вновь создаваемых ключей; в дополнение к этому буферы, на которые указывают поля *pIVClient* и *pIVServer* структуры **CK_WTLS_KEY_MAT_OUT**, будут содержать выданные в них векторы инициализации IV (если векторы инициализации IV запрашивались автором запроса). Следовательно, эти два поля должны указывать на буферы, имеющие достаточно места для размещения любого IV, который будет выдан.

Данный механизм дистанцируется от остальных механизмов деривации ключей в Cryptoki по выдаваемой им информации. Для большинства механизмов деривации ключа **C_DeriveKey** выдает в качестве успешного результата завершения операции один идентификатор ключа. Однако, поскольку механизм **CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE** выдает все свои идентификаторы ключа в структуру **CK_WTLS_KEY_MAT_OUT**, на которую указывает структура **CK_WTLS_KEY_MAT_PARAMS**, заданная как параметр механизма, параметр *phKey*, передаваемый в **C_DeriveKey,** не нужен и его содержимое должно быть NULL_PTR.

Если запрос данного механизма к **C_DeriveKey** не удовлетворяется, тогда на носителе не будет создан *ни один* из двух ключей.

12.33.8 Деривация клиентского ключа и MAC

Деривация клиентского ключа, MAC и вектора инициализации IV в WTLS, обозначенная как **CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE**, представляет собой механизм, используемый для выведения соответствующего криптографического ключевого материала, используемого шифрсхемой, из данных секретного мастер-ключа и случайных данных. Данный механизм выдает идентификаторы для ключей, генерируемых в данном процессе, а также создаваемые векторы инициализации IV.

Данный механизм имеет параметр структуры **CK_WTLS_KEY_MAT_PARAMS**, который позволяет передавать тип механизма списка, который надлежит использовать, случайные данные, характеристики криптографического материала для заданной шифрсхемы, а также передавать указатель на структуру, которая получает сгенерированные идентификаторы и векторы инициализации.

Данный механизм вносит вклад в создание на носителе двух отдельных ключей и выдает автору запроса один вектор инициализации IV (если вектор инициализации IV запрашивается автором запроса). Всем этим ключам придается класс объекта **CKO_SECRET_KEY**.

Ключу MAC (client_write_MAC_secret) всегда придается тип **CKK_GENERIC_SECRET**. Он обозначается флагом как пригодный для операций электронной подписи, верификации и деривации.

Другому ключу (client_write_key) придается тип в соответствии с информацией, обнаруженной в шаблоне, посылаемом вместе с данным механизмом в ходе выполнения функции запроса **C_DeriveKey**. По умолчанию он обозначается флагом как пригодный для операций шифрования, расшифровки и деривации.

Вектор инициализации (client write IV) будет генерироваться и выдаваться, если поле *ulIVSizeInBits* в структуре **CK_WTLS_KEY_MAT_PARAMS** имеет не нулевое значение. Если вектор инициализации IV генерируется, его длина в битах будет согласовываться с величиной, содержащейся в поле *ulIVSizeInBits*.

Оба ключа наследуют у базового ключа значения его атрибутов **CKA_SENSITIVE**, **CKA_ALWAYS_ SENSITIVE**, **CKA_EXTRACTABLE** и **CKA_NEVER_EXTRACTABLE**. Шаблон, предоставляемый для **C_DeriveKey**, может не определять значения для любого из этих атрибутов, которые отличаются от значений базового ключа.

Следует отметить, что структура **CK_WTLS_KEY_MAT_OUT**, на которую указывает поле *pReturnedKeyMaterial* структуры **CK_WTLS_KEY_MAT_PARAMS**, будет модифицирована запросом **C_DeriveKey**. В частности, два поля идентификаторов ключей в структуре **CK_WTLS_KEY_MAT_OUT** будут модифицированы таким образом, чтобы данная структура содержала идентификаторы для вновь создаваемых ключей; в дополнение к этому буферы, на которые указывают поля *pIVClient* и *pIVServer* структуры **CK_WTLS_KEY_MAT_OUT**, будут содержать выданные в них векторы инициализации IV (если векторы инициализации IV запрашивались автором запроса). Следовательно, эти два поля должны указывать на буферы, имеющие достаточно места для размещения любого вектор инициализации IV, который будет выдан.

Данный механизм дистанцируется от остальных механизмов деривации ключей в Cryptoki по выдаваемой им информации. Для большинства механизмов деривации ключа **C_DeriveKey** выдает в качестве успешного результата завершения операции один идентификатор ключа. Однако, поскольку механизм **CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE** выдает все свои идентификаторы ключа в структуру **CK_WTLS_KEY_MAT_OUT**, на которую указывает структура **CK_WTLS_KEY_MAT_PARAMS**, заданная как параметр механизма, параметр *phKey*, передаваемый в **C_DeriveKey,** не нужен и его содержимое должно быть NULL_PTR.

Если запрос данного механизма к **C_DeriveKey** не удовлетворяется, тогда на носителе не будет создан *ни один* из двух ключей.

12.34 Разнообразные простые механизмы деривации ключей

12.34.1 Определения

Механизмы:

```
CKM_CONCATENATE_BASE_AND_DATA
CKM_CONCATENATE_DATA_AND_BASE
CKM_XOR_BASE_AND_DATA
CKM_EXTRACT_KEY_FROM_KEY
CKM_CONCATENATE_BASE_AND_KEY
```

12.34.2 Параметры для разнообразных простых механизмов деривации ключей

♦ **CK_KEY_DERIVATION_STRING_DATA; CK_KEY_DERIVATION_STRING_DATA_PTR**

CK_KEY_DERIVATION_STRING_DATA предоставляет параметры для механизмов **CKM_CONCATENATE_ BASE_AND_DATA**, **CKM_CONCATENATE_DATA_AND_BASE** и **CKM_XOR_BASE_AND_ DATA**. Данный механизм задается следующим образом:

```
typedef struct CK_KEY_DERIVATION_STRING_DATA {
  CK_BYTE_PTR pData;
  CK_ULONG ulLen;
} CK_KEY_DERIVATION_STRING_DATA;
```

Поля данной структуры имеют следующие значения:

pData указатель на байтовую строку;

ulLen длина байтовой строки.

CK_KEY_DERIVATION_STRING_DATA_PTR является указателем на **CK_KEY_DERIVATION_ STRING_DATA**.

♦ **CK_EXTRACT_PARAMS; CK_EXTRACT_PARAMS_PTR**

CK_KEY_EXTRACT_PARAMS предоставляет параметр для механизма **CKM_EXTRACT_KEY_ FROM_KEY**. Данный механизм устанавливает, какой бит базового ключа должен использоваться как первый бит выводимого ключа. Данный механизм задается следующим образом:

```
typedef CK_ULONG CK_EXTRACT_PARAMS;
```

CK_EXTRACT_PARAMS_PTR является указателем на **CK_EXTRACT_PARAMS**.

12.34.3 Соединение базового ключа с другим ключом

Данный механизм, обозначенный как **CKM_CONCATENATE_BASE_AND_KEY**, выводит ключ из соединения двух существующих секретных ключей. Эти два ключа задаются идентификаторами; значения заданных ключей соединяются воедино в буфере.

Данный механизм принимает параметр **CK_OBJECT_HANDLE**. Данный идентификатор производит информацию о значении ключа, которая прилагается к информации о значении базового ключа (базовый ключ – это ключ, идентификатор которого предоставляется в качестве аргумента к механизму **C_DeriveKey**).

Например, если значение базового ключа `0x01234567`, а значение другого ключа `0x89ABCDEF`, тогда значение выведенного ключа будет браться из буфера, содержащего строку `0x0123456789ABCDEF`.

- Если в шаблоне не предоставляются сведения о длине или типе ключа, тогда ключ, производимый данным механизмом, будет общим секретным ключом. Его длина будет равна сумме длин значений двух исходных ключей.
- Если в шаблоне не предоставляются сведения о типе ключа, но предоставляются сведения о его длине, тогда ключ, производимый данным механизмом, будет общим секретным ключом заданной длины.
- Если в шаблоне не предоставляются сведения о длине ключа, но предоставляются сведения о его типе, тогда данный тип ключа должен иметь верно заданную длину. Если это так, тогда ключ, производимый данным механизмом, будет того типа, который определен в шаблоне. Если это не так, будет выдано сообщение об ошибке.
- Если в шаблоне предоставлены сведения и о типе, и о длине ключа, то данная длина должна быть совместима с данным типом. Ключ, производимый данным механизмом, будет заданного типа и длины.

Если при помощи данного механизма будет выводиться ключ DES, DES2, DES3 или CDMF, биты четности данного ключа будут расположены верно.

Если запрошенный тип ключа требует больше байт, чем можно получить соединением значений двух исходных ключей, будет сгенерировано сообщение об ошибке.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Если у каждого из двух исходных ключей атрибут **CKA_SENSITIVE** установлен на CK_TRUE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_SENSITIVE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.
- Аналогично, если у каждого из двух исходных ключей атрибут **CKA_EXTRACTABLE** установлен на CK_FALSE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_EXTRACTABLE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.
- Атрибут выведенного ключа **CKA_ALWAYS_SENSITIVE** устанавливается на CK_TRUE тогда и только тогда, когда у обоих исходных ключей атрибуты **CKA_ALWAYS_SENSITIVE** установлены на CK_TRUE.
- Аналогично, атрибут выведенного ключа **CKA_NEVER_EXTRACTABLE** устанавливается на CK_TRUE тогда и только тогда, когда у обоих исходных ключей атрибуты **CKA_NEVER_EXTRACTABLE** установлены на CK_TRUE.

12.34.4 Соединение базового ключа и данных

Данный механизм, обозначенный как **CKM_CONCATENATE_BASE_AND_DATA**, выводит секретный ключ посредством присоединения данных к замыкающей части заданного секретного ключа.

Данный механизм принимает параметр структуры **CK_KEY_DERIVATION_STRING_DATA**, которая задает длину и значение данных, которые будут приданы базовому ключу для деривации другого ключа.

Например, если значение базового ключа `0x01234567`, а значение данных `0x89ABCDEF`, тогда значение выводимого ключа будет браться из буфера, содержащего строку `0x0123456789ABCDEF`.

- Если в шаблоне не предоставляются сведения о длине или типе ключа, тогда ключ, производимый данным механизмом, будет общим секретным ключом. Его длина будет равна сумме длин значения и данных двух исходных ключей.
- Если в шаблоне не предоставляются сведения о типе ключа, но предоставляются сведения о его длине, тогда ключ, производимый данным механизмом, будет общим секретным ключом заданной длины.
- Если в шаблоне не предоставляются сведения о длине ключа, но предоставляются сведения о его типе, тогда данный тип ключа должен иметь верно заданную длину. Если это так, тогда ключ,

производимый данным механизмом, будет того типа, который определен в шаблоне. Если это не так, будет выдано сообщение об ошибке.

- Если в шаблоне предоставлены сведения и о типе, и о длине ключа, то данная длина должна быть совместима с данным типом. Ключ, производимый данным механизмом, будет заданного типа и длины.

Если при помощи данного механизма будет выводиться ключ DES, DES2, DES3 или CDMF, биты четности данного ключа будут расположены верно.

Если запрошенный тип ключа требует больше байт, чем можно получить соединением значения исходного ключа и данных, будет сгенерировано сообщение об ошибке.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Если у базового ключа атрибут **CKA_SENSITIVE** установлен на CK_TRUE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_SENSITIVE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.
- Аналогично, если у базового ключа атрибут **CKA_EXTRACTABLE** установлен на CK_FALSE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_EXTRACTABLE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.
- Атрибут выведенного ключа **CKA_ALWAYS_SENSITIVE** устанавливается на CK_TRUE тогда и только тогда, когда у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** установлен на CK_TRUE.
- Аналогично, атрибут выведенного ключа **CKA_NEVER_EXTRACTABLE** устанавливается на CK_TRUE тогда и только тогда, когда у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** установлен на CK_TRUE.

13.34.5 Соединение данных и базового ключа

Данный механизм, обозначенный как **CKM_CONCATENATE_DATA_AND_BASE**, выводит секретный ключ посредством присоединения данных перед начальной частью заданного секретного ключа.

Данный механизм принимает параметр структуры **CK_KEY_DERIVATION_STRING_DATA**, которая задает длину и значение данных, которые будут присоединены перед передней частью базового ключа для деривации другого ключа.

Например, если значение базового ключа `0x01234567`, а значение данных `0x89ABCDEF`, тогда значение выводимого ключа будет браться из буфера, содержащего строку `0x89ABCDEF01234567`.

- Если в шаблоне не предоставляются сведения о длине или типе ключа, тогда ключ, производимый данным механизмом, будет общим секретным ключом. Его длина будет равна сумме длин данных и значения двух исходных ключей.
- Если в шаблоне не предоставляются сведения о типе ключа, но предоставляются сведения о его длине, тогда ключ, производимый данным механизмом, будет общим секретным ключом заданной длины.
- Если в шаблоне не предоставляются сведения о длине ключа, но предоставляются сведения о его типе, тогда данный тип ключа должен иметь верно заданную длину. Если это так, тогда ключ, производимый данным механизмом, будет того типа, который определен в шаблоне. Если это не так, будет выдано сообщение об ошибке.
- Если в шаблоне предоставлены сведения и о типе, и о длине ключа, то данная длина должна быть совместима с данным типом. Ключ, производимый данным механизмом, будет заданного типа и длины.

Если при помощи данного механизма будет выводиться ключ DES, DES2, DES3 или CDMF, биты четности данного ключа будут расположены верно.

Если запрошенный тип ключа требует больше байт, чем можно получить соединением данных и значения исходного ключа, будет сгенерировано сообщение об ошибке.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Если у базового ключа атрибут **CKA_SENSITIVE** установлен на CK_TRUE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_SENSITIVE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.

- Аналогично, если у базового ключа атрибут **CKA_EXTRACTABLE** установлен на CK_FALSE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута

CKA_EXTRACTABLE выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.

- Атрибут выведенного ключа **CKA_ALWAYS_SENSITIVE** устанавливается на CK_TRUE тогда и только тогда, когда у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** установлен на CK_TRUE.
- Аналогично, атрибут выведенного ключа **CKA_NEVER_EXTRACTABLE** устанавливается на CK_TRUE тогда и только тогда, когда у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** установлен на CK_TRUE.

12.34.6 Сложение ключа и данных по модулю 2 (XOR)

Деривация ключа посредством сложения по модулю 2 (XOR), обозначенная как **CKM_XOR_BASE_AND_DATA**, представляет собой механизм, который предоставляет возможность деривации секретного ключа через побитовое сложение по модулю 2 (XOR) ключа, на который указывает идентификатор базового ключа, и некоторых данных.

Данный механизм принимает параметр структуры **CK_KEY_DERIVATION_STRING_DATA**, который определяет данные, с которыми надлежит складывать по модулю 2 (XOR) значение исходного ключа.

Например, если значение исходного ключа `0x01234567`, а значение данных `0x89ABCDEF`, тогда значение выводимого ключа будет взято из буфера, содержащего строку `0x88888888`.

- Если в шаблоне не предоставляются сведения о длине или типе ключа, тогда ключ, производимый данным механизмом, будет общим секретным ключом. Его длина будет эквивалентна минимуму из длин даты и значения исходного ключа.
- Если в шаблоне не предоставляются сведения о типе ключа, но предоставляются сведения о его длине, тогда ключ, производимый данным механизмом, будет общим секретным ключом заданной длины.
- Если в шаблоне не предоставляются сведения о длине ключа, но предоставляются сведения о его типе, тогда данный тип ключа должен иметь верно заданную длину. Если это так, тогда ключ, производимый данным механизмом, будет того типа, который определен в шаблоне. Если это не так, будет выдано сообщение об ошибке.
- Если в шаблоне предоставлены сведения и о типе, и о длине ключа, то данная длина должна быть совместима с данным типом. Ключ, производимый данным механизмом, будет заданного типа и длины.

Если при помощи данного механизма будет выводиться ключ DES, DES2, DES3 или CDMF, биты четности данного ключа будут расположены верно.

Если запрашиваемый тип ключа требует больше байт, чем можно получить, взяв наиболее короткое значение из значений данных и исходного ключа, будет сгенерировано сообщение об ошибке.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Если у базового ключа атрибут **CKA_SENSITIVE** установлен на CK_TRUE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_SENSITIVE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.
- Аналогично, если у базового ключа атрибут **CKA_EXTRACTABLE** установлен на CK_FALSE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_EXTRACTABLE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.
- Атрибут выведенного ключа **CKA_ALWAYS_SENSITIVE** устанавливается на CK_TRUE тогда и только тогда, когда у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** установлен на CK_TRUE.
- Аналогично, атрибут выведенного ключа **CKA_NEVER_EXTRACTABLE** устанавливается на CK_TRUE тогда и только тогда, когда у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** установлен на CK_TRUE.

12.34.7 Извлечение одного ключа из другого ключа

Извлечение одного ключа из другого ключа, обозначенное как **CKM_EXTRACT_KEY_FROM_KEY**, представляет собой механизм, который предоставляет возможность создания одного секретного ключа из бит другого секретного ключа.

Данный механизм имеет параметр CK_EXTRACT_PARAMS, который указывает, который бит исходного ключа должен быть использован как первый бит вновь выводимого ключа.

Даем пример того, как данный механизм работает. Предположим, носитель имеет секретный ключ с 4-байтным значением 0x329F84A9. Мы можем вывести 2-байтный секретный ключ из этого ключа, начиная с битовой позиции 21 (т. е. значение данного параметра для механизма

CKM_EXTRACT_KEY_FROM_KEY – 21).

1. Пишем значение ключа в двоичной форме: 0011 0010 1001 1111 1000 0100 1010 1001. Рассматриваем данную двоичную строку как содержащую 32 бита ключа, помеченных b0, b1, …, b31.

2. Затем извлекаем 16 последовательных бит (т. е. 2 байта) из этой двоичной строки, начиная с бита b21. Получаем двоичную строку 1001 0101 0010 0110.

3. Таким образом, значение нового ключа 0x9526.

Следует отметить, что при создании значения выводимого ключа допускается зацикливать (to wrap around) конец двоичной строки, представляющей значение исходного ключа.

Если исходный ключ, используемый в данном процессе, является чувствительным (sensitive), тогда выводимый ключ должен также быть чувствительным для последующего процесса деривации.

- Если в шаблоне не предоставляются сведения о длине или типе ключа, тогда выдается сообщение об ошибке.
- Если в шаблоне не предоставляются сведения о типе ключа, но предоставляются сведения о его длине, тогда ключ, производимый данным механизмом, будет общим секретным ключом заданной длины.
- Если в шаблоне не предоставляются сведения о длине ключа, но предоставляются сведения о его типе, тогда данный тип ключа должен иметь верно заданную длину. Если это так, тогда ключ, производимый данным механизмом, будет того типа, который определен в шаблоне. Если это не так, будет выдано сообщение об ошибке.
- Если в шаблоне предоставлены сведения и о типе, и о длине ключа, то данная длина должна быть совместима с данным типом. Ключ, производимый данным механизмом, будет заданного типа и длины.

Если при помощи данного механизма будет выводиться ключ DES, DES2, DES3 или CDMF, биты четности данного ключа будут расположены верно.

Если запрашиваемый тип ключа требует больше байт, чем имеет исходный ключ, будет сгенерировано сообщение об ошибке.

Данному механизму присущи следующие правила по чувствительности и извлекаемости ключа:

- Если у базового ключа атрибут **CKA_SENSITIVE** установлен на CK_TRUE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_SENSITIVE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.
- Аналогично, если у базового ключа атрибут **CKA_EXTRACTABLE** установлен на CK_FALSE, таким же будет и состояние выведенного ключа. Если этого нет, тогда состояние атрибута **CKA_ EXTRACTABLE** выведенного ключа будет установлено на основании предоставленного шаблона либо он примет значение по умолчанию.
- Атрибут выведенного ключа **CKA_ALWAYS_SENSITIVE** устанавливается на CK_TRUE тогда и только тогда, когда у базового ключа атрибут **CKA_ALWAYS_SENSITIVE** установлен на CK_TRUE.
- Аналогично, атрибут выведенного ключа **CKA_NEVER_EXTRACTABLE** устанавливается на CK_TRUE тогда и только тогда, когда у базового ключа атрибут **CKA_NEVER_EXTRACTABLE** установлен на CK_TRUE.

12.35 CMS

12.35.1 Определения

Механизмы:

`CKM_CMS_SIG`

12.35.2 Объекты механизмов электронной подписи CMS

Данные объекты предоставляют информацию, относящуюся к механизму CKM_CMS_SIG. Атрибуты объекта механизма CKM_CMS_SIG представляют собой информацию о поддерживаемых в носителе атрибутах электронной подписи CMS. Они представлены только на носителях, поддерживающих механизм **CKM_CMS_SIG**, но на этих носителях они присутствовать должны.

Таблица 145 – Атрибуты объекта механизма электронной подписи CMS

Атрибут	Тип данных	Значение
CKA_REQUIRED_CMS_ATTRIBUTES	Совокупность байтов	Атрибуты, которые носитель будет всегда включать в набор атрибутов, подписываемых посредством CMS
CKA_DEFAULT_CMS_ATTRIBUTES	Совокупность байтов	Атрибуты, которые носитель будет включать в набор атрибутов, подписываемых посредством CMS, в отсутствие каких бы то ни было атрибутов, заданных приложением
CKA_SUPPORTED_CMS_ATTRIBUTES	Совокупность байтов	Атрибуты, которые носитель может включать в набор атрибутов, подписываемых посредством CMS, по запросу приложения

Содержанием каждой совокупности байтов будет закодированный посредством DER список атрибутов CMS **Attributes** с необязательными сопутствующими значениями. Любые атрибуты в данном списке должны быть идентифицированы при помощи своего идентификатора объекта, а любые значения должны быть закодированы посредством DER. Список атрибутов задается в ASN.1 следующим образом:

```
Attributes ::= SET SIZE (1..MAX) OF Attribute
Attribute ::= SEQUENCE {
    attrType OBJECT IDENTIFIER,
    attrValues SET OF ANY DEFINED BY OBJECT IDENTIFIER OPTIONAL
}
```

Заказчик может не устанавливать никаких атрибутов.

12.35.3 Параметры механизма CMS

- **CK_CMS_SIG_PARAMS, CK_CMS_SIG_PARAMS_PTR**

CK_CMS_SIG_PARAMS представляет собой структуру, которая предоставляет параметры механизму **CKM_CMS_SIG**. Данная структура задается следующим образом:

```
typedef struct CK_CMS_SIG_PARAMS {
CK_OBJECT_HANDLE          certificateHandle;
CK_MECHANISM_PTR          pSigningMechanism;
CK_MECHANISM_PTR          pDigestMechanism;
CK_UTF8CHAR_PTR           pContentType;
CK_BYTE_PTR               pRequestedAttributes;
CK_ULONG            ulRequestedAttributesLen;
CK_BYTE_PTR               pRequiredAttributes;
CK_ULONG            ulRequiredAttributesLen;
} CK_CMS_SIG_PARAMS;
```

Поля данной структуры имеют следующие значения:

certificateHandle идентификатор объекта для сертификата, ассоциированного с ключом, используемым для электронной подписи. Носитель может использовать информацию из этого сертификата, чтобы идентифицировать подписывающего в значении результата **SignerInfo**. Поле *Certificate-Handle* может иметь значение NULL_PTR, если данный сертификат недоступен как объект PKCS #11 или если вызывающее приложение полностью оставляет носителю выбор сертификата;

pSigningMechanism механизм, который надлежит использовать при постановке электронной подписи на созданном значении CMS **SignedAttributes**. Например **CKM_SHA1_RSA_PKCS**;

pDigestMechanism механизм, который надлежит использовать при получении списка данных. Значение должно быть NULL_PTR перед использованием механизма списка из параметра *pSigningMechanism*;

pContentType строка, заканчивающаяся значением NULL и обозначающая тип по содержанию MIME (MIME Content-type) сообщения, подлежащего подписанию; или значение NULL_PTR, если сообщение является объектом MIME (который может быть проанализирован носителем, чтобы определить его тип по содержанию MIME, если это требует-

	ся). Нужно использовать значение «`application/octet-stream`», если тип MIME сообщения неизвестен или не задан. Следует отметить, что строка *pContentType* должна соответствовать синтаксису, определенному в RFC 2045, т. е. должны присутствовать любые параметры, необходимые для верного представления содержание носителем (такие, например, как незадаваемый по умолчанию параметр «`charset`»). Представляя содержание, носитель должен следовать правилам и процедурам, определенным в RFC 2045;
pRequestedAttributes	указатель на закодированный посредством DER список атрибутов CMS **Attributes**, включения которого в подписываемые атрибуты требует автор запроса. Носитель может свободно проигнорировать включение этого показателя в подписанные атрибуты. Носитель может полностью игнорировать данный список или модифицировать любые представленные значения;
ulRequestedAttributesLen	длина в байтах значения, на которое указывает полет *pRequested-Attributes*;
pRequiredAttributes	указатель на закодированный посредством DER список атрибутов CMS **Attributes** (с сопутствующими значениями), которые требуется включить в результирующие подписанные атрибуты. Носитель не должен модифицировать никакие из поставленных значений. Если носитель не поддерживает один или более атрибутов или не принимает предоставленные значения, операция постановки подписи проведена не будет. Носитель будет использовать при подписании свои собственные атрибуты по умолчанию, если оба поля *pRequested-Attributes* и *pRequiredAttributes* будут установлены на NULL_PTR;
ulRequiredAttributesLen	длина в байтах значения, на которое указывает поле *pRequired-Attributes*.

12.35.4 Подписи CMS

Механизм CMS, обозначенный как **CKM_CMS_SIG**, представляет собой многоцелевой механизм, основанный на структурах, определенных в PKCS #7 и RFC 2630. Данный механизм поддерживает односторонние и многосторонние процедуры постановки электронной подписи с восстановлением сообщения и без него. Этот механизм предназначен для использования, например, с устройствами PTD (см. MeT-PTD) или с другими носителями, обладающими соответствующими возможностями. Носитель будет составлять значение CMS **SignedAttributes** и вычислять из этого значения подпись. Решение по содержанию значения **SignedAttributes** принимает носитель, однако автор запроса может предложить некоторые атрибуты в параметр *pRequestedAttributes*. Автор запроса может также потребовать, чтобы некоторые атрибуты находились в параметрах *pRequiredAttributes*. Электронная подпись вычис-ляется в соответствии с параметром *pSigningMechanism*.

Когда данный механизм используется в успешных запросах к **C_Sign** или **C_SignFinal**, выдаваемое значение *pSignature* будет указывать на закодированное посредством DER значение типа **SignerInfo**. **SignerInfo** определяется в ASN.1 следующим образом (для полного определения всех полей и типов см. RFC 2630):

```
SignerInfo ::= SEQUENCE {
    version CMSVersion,
    sid SignerIdentifier,
    digestAlgorithm DigestAlgorithmIdentifier,
    signedAttrs [0] IMPLICIT SignedAttributes OPTIONAL,
    signatureAlgorithm SignatureAlgorithmIdentifier,
    signature SignatureValue,
    unsignedAttrs [1] IMPLICIT UnsignedAttributes OPTIONAL }
```

Параметр *certificateHandle*r, когда он задан, помогает носителю заполнить поле **sid** значения **SignerInfo** value. Если параметр *certificateHandle* принимает значение NULL_PTR, выбор ссылки на подходящий сертификат в **SignerInfo** остается за носителем (носитель может, например, проконсультироваться с пользователем).

Данный механизм не должен использоваться в запросах к **C_Verify** или **C_VerifyFinal** (вместо этого надлежит использовать механизм *pSigningMechanism*).

Чтобы приложение разобралось, какие атрибуты поддерживаются носителем, какие атрибуты будут использоваться по умолчанию, какие атрибуты будут добавляться всегда, данное приложение должно проанализировать содержание объекта **CKH_CMS_ATTRIBUTES,** имеющего информацию о характеристиках оборудования.

Для поля *pRequiredAttributes* носитель может оказаться перед необходимостью связаться с пользователем, чтобы выяснить, принимать или нет предложенное значение. Носитель никогда не должен принимать никакого из предложенных значений атрибутов без своего рода подтверждения от своего владельца (но это может быть осуществлено, например, через установки конфигурации или политики, а не через прямое взаимодействие). Если пользователь отвергает предложенные значения или запрос подписи как таковой, будет выдано значение CKR_FUNCTION_REJECTED.

Когда возможно, при генерировании подписей, совместимых с CMS-приложениями, необходимо использовать механизм **CKM_CMS_SIG**, а не механизмы низшего уровня, как, например, **CKM_SHA1_ RSA_PKCS**. Это особенно верно, когда подписи должны создаваться на содержании, которое носитель может предъявить пользователю. Исключение могут составлять те случаи, когда носитель не поддерживает определенный атрибут процедуры подписания. Следует, однако, отметить, что носитель может отказаться от использования определенного ключа для подписи, если не известно содержание, подлежащее подписанию (т. е. используется механизм **CKM_CMS_SIG**).

Когда носитель не имеет возможностей представления содержания, приложение, знакомое с PKCS #11, может избежать отправления на носитель целого сообщения посредством выбора подходящего механизма подписания (например, **CKM_RSA_PKCS**) в качестве значения *pSigningMechanism* в структуре **CKM_CMS_SIG_PARAMS** и через получение списка сообщения перед передачей его на носитель.

Приложения, знакомые с PKCS #11 при использовании ими носителей с возможностями представления содержания, должны пытаться обеспечить подписание сообщений носителем в формате, который давал бы носителю возможность представлять сообщение пользователю. Носители, которые получают многосторонние сообщения типа MIME, позволяющие представлять лишь отдельные их части, могут не осуществить процедуру подписания с выдачей значения **CKR_DATA_INVALID**, однако они могут также выбрать добавление атрибута процедуры подписания, указывающего, какие части сообщения есть возможность представить.

12.36 Blowfish

Blowfish – блочный шифр с секретным ключом. Данный шифр представляет собой сеть Фейстеля, повторяющую простую функцию шифрования 16 раз. Размер блока – 64 бита, ключ может быть любой длины до 448 бит. Хотя имеет место сложная инициализационная фаза, которую требуется пройти перед тем, как начнется любое шифрование, само шифрование данных весьма эффективно на больших микропроцессорах (см. информацию по адресу http://www.counterpane.com/bfsverlag.html).

12.36.1 Определения

Данный раздел определяет тип ключа «CKK_BLOWFISH» как тип CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_BLOWFISH_KEY_GEN
CKM_BLOWFISH_CBC
```

12.36.2 Объекты секретного ключа BLOWFISH

Объекты секретного ключа Blowfish (класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_ BLOWFISH**) содержат ключи Blowfish. В таблице 146 определены атрибуты объекта секретного ключа в дополнение к общим атрибутам, определенным для объектов данного класса:

Таблица 146 – Объект секретного ключа BLOWFISH

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Значение ключа может быть любой длины до 448 бит. Длина в битах ограничена совокупностью байтов
CKA_VALUE_LEN [2, 3]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Ниже приводится образец шаблона для создания объекта секретного ключа Blowfish:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
```

```
CK_KEY_TYPE keyType = CKK_BLOWFISH;
CK_UTF8CHAR label[] = “Объект секретного ключа Blowfish”;
CK_BYTE value[16] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.36.3 Генерация ключа Blowfish

Механизм генерации ключа Blowfish, обозначенный как **CKM_BLOWFISH_KEY_GEN**, представляет собой механизм генерации ключа Blowfish.

Данный механизм не имеет параметра.

Данный механизм генерирует ключи Blowfish с определенной длиной, как задано в атрибуте **CKA_VALUE_LEN** шаблона для данного ключа.

Данный механизм добавляет к новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE**. Другие атрибуты, поддерживаемые данным типом ключа (в частности, флаги, показывающие, какие функции данный ключ поддерживает), могут быть заданы в шаблоне данного ключа, либо им могут придаваться некоторые значения по умолчанию.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа в байтах.

12.36.4 Blowfish-CBC

Механизм Blowfish-CBC, обозначенный как **CKM_BLOWFISH_CBC**, является механизмом для одностороннего и многостороннего шифрования и расшифровки, упаковки и распаковки ключей.

Данный механизм имеет параметр 16-байтного вектора инициализации.

12.37 Twofish

- 128-битный блок
- 128-, 192- или 256-битный ключ
- 16 этапов
- Работает во всех стандартных режимах
- Эффективная клиентская настройка ключа на больших микропроцессорах
- Эффективная работа со смарт-картами
- Эффективная работа аппаратной части
- Методика широко подвергалась криптоаналитической экспертизе
- Методика не защищена патентами
- Методика не защищена правами собственности
- Предоставляется бесплатно

См. http://www.counterpane.com/twofish-brief.html

12.37.1 Определения

Данный раздел определяет тип ключа CKK_TWOFISH как тип CK_KEY_TYPE, используемый в атрибуте CKA_KEY_TYPE ключевых объектов.

Механизмы:

```
CKM_TWOFISH_KEY_GEN
CKM_TWOFISH_CBC
```

12.37.2 Объекты секретного ключа Twofish

Объекты секретного ключа Twofish (класс объекта – **CKO_SECRET_KEY**, тип ключа – **CKK_TWOFISH**) содержат ключи Twofish. В таблице 147 определены атрибуты объекта секретного ключа в дополнение к общим атрибутам, определенным для объектов данного класса:

Таблица 147 – Объект секретного ключа Twofish

Атрибут	Тип данных	Значение

Атрибут	Тип данных	Значение
CKA_VALUE [1, 4, 6, 7]	Совокупность байтов	Длина значения ключа 128, 192 или 256 бит
CKA_VALUE_LEN [2, 3]	CK_ULONG	Длина значения ключа в байтах
См. сноски в таблице 15.		

Ниже приводится образец шаблона для создания объекта секретного ключа TWOFISH:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_TWOFISH;
CK_UTF8CHAR label[] = “Объект секретного ключа TWOFISH”;
CK_BYTE value[16] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

12.37.3 Генерация ключа Twofish

Механизм генерации ключа Twofish, обозначенный как **CKM_TWOFISH_KEY_GEN**, является механизмом генерации ключа Twofish.

Данный механизм не имеет параметра.

Данный механизм генерирует ключи Blowfish с определенной длиной, как определено в атрибуте **CKA_VALUE_LEN** шаблона данного ключа.

Данный механизм добавляет к новому ключу атрибуты **CKA_CLASS**, **CKA_KEY_TYPE** и **CKA_VALUE**. Другие атрибуты, поддерживаемые данным типом ключа (в частности, флаги, показывающие, какие функции данный ключ поддерживает), могут быть заданы в шаблоне данного ключа, либо им могут быть приданы некоторые значения по умолчанию.

Для данного механизма поля *ulMinKeySize* и *ulMaxKeySize* структуры **CK_MECHANISM_INFO** определяют поддерживаемый диапазон размеров ключа в байтах.

12.37.4 Twofish-CBC

Механизм Twofish-CBC, обозначенный как **CKM_TWOFISH_CBC**, является механизмом для одностороннего и многостороннего шифрования и расшифровки, упаковки и распаковки ключей.

Данный механизм имеет параметр 16-байтного вектора инициализации.

13 Рекомендации и напоминания по Cryptoki

В настоящем раделе рассматриваются повторно и/или более подробно некоторые моменты, связанные с тем, как работает Cryptoki.

13.1 Операции, сеансы и цепочки

В Cryptoki имеется несколько различных типов операций, которые могут быть «активными» в сеансе. Активная операция – именно та, которая принимает для исполнения запрос на более чем одну функцию Cryptoki. Типы активных операций – поиск обекта, шифрование, расшифровка, получение списка сообщения, подпись с приложением, подпись с восстановлением, верификация с приложением, а также верификация с восстановлением.

Определенный сеанс может иметь одновременно активными 0, 1 или 2 операции. Он может иметь активными одновременно только 2 операции, если носитель это поддерживает; более того, две эти операции должны представлять собой одну из четырех следующих пар операций: получение списка и шифрование; расшифровку и получение списка; постановку подписи и шифрование; расшифровку и верификацию.

Если приложение пытается инициализировать операцию (сделать ее активной) в сеансе, но это не может быть осуществлено из-за наличия другой активной операции (других активных операций),

данное приложение получает значение, свидетельствующее об ошибке CKR_OPERATION_ACTIVE. Это значение, свидетельствующее об ошибке, может также быть получено, если сеанс имеет активную операцию, а приложение пытается использовать данный сеанс для выполнения любой из различных операций, которые не становятся активными, но требуют выполнения криптографических процедур, таких как использование генератора случайных чисел носителя либо генерация/упаковка/распаковка/деривация ключа.

Чтобы отказаться от активной операции, приложению может понадобиться завершить операцию и отказаться от результата. Закрытие сеанса будет также иметь этот эффект. Напротив, библиотека может позволить завершение приложением активных операций просто посредством инициализации некоторой другой операции. В этом случае не будет выдаваться сообщение CKR_OPERATION_ ACTIVE, но предыдущая активная операция использоваться не будет.

Различные цепочки в работе приложения никогда не будут пользоваться общими сеансами, пока они не станут предельно внимательными относительно того, чтобы не осуществлять функциональных запросов в одно и то же время. Это верно даже для случая, когда библиотека Cryptoki была бы инициализирована приведением в режим готовности к обеспечению защиты от цепочек операций.

13.2 Поведение при работе с приложениями многостороннего доступа

Когда многосторонние приложения или многосторонние цепочки внутри одного приложения осуществляют доступ к набору общих объектов, становится важным вопрос защиты объекта. В особенности это касается случаев, когда приложение A активизирует операцию с использованием объекта O, а приложение B пытается удалить O до того, как приложение A завершило действие. К сожалению, различия в возможностях устройства делают исчерпывающее описание работы с устройством невозможным. Здесь представлены общие рекомендации по действиям, направленным на защиту объекта.

Всегда, когда это возможно, удаление объекта в одном из приложений не должно вести к тому, чтобы этот объект становился недоступным для другого приложения или цепочки действий, использующих данный объект в активной операции, до того, как эта операция не завершится. Например, приложение A начало процедуру постановки подписи с секретным ключом P, а приложение B пытается удалить P, в то время как процедура постановки подписи продолжается. В этом случае может произойти одно из двух. Объект удаляется с устройства, однако операцию можно завершить, поскольку данная операция использует временную копию объекта, или операция удаления блокируется до того времени, когда операция постановки подписи не будет завершена. Если ни одно из этих действий не может быть поддержано в действующей реализации аппаратно-программного комплекса, тогда приложению A может быть выдан код ошибки CKR_OBJECT_HANDLE_INVALID, что указывает на факт удаления ключа, использовавшегося данным приложением в его активной операции.

Когда это возможно, изменение значения атрибута объекта должно влиять на поведение активных операций в других приложениях или цепочках. Если это не может быть поддержано приложением, тогда приложению с активной операцией будет выдан соответствующий код ошибки, указывающий причину сбоя.

13.3 Объекты, атрибуты и шаблоны

Любая функция Cryptoki, требующая шаблона для объекта, нуждается в шаблоне, чтобы задать – в явной либо скрытой форме – какие-то атрибуты, которые нигде не определены. Если шаблон дает подробное описание определенного атрибута более одного раза, функция может выдать сообщение CKR_TEMPLATE_INVALID либо она может выбрать определенное значение атрибута из тех, что заданы, и использовать данное значение. В любом случае атрибуты объекта всегда однозначны.

13.4 Подписание с восстановлением

Подписание с восстановлением представляет собой общую альтернативу обычным электронным подписям («подписание с приложением»), которые поддерживаются определенными механизмами. Напомним, что для обычных электронных подписей подпись сообщения вычисляется как некоторая функция сообщения и секретного ключа обладателя подписи; данная подпись может затем использоваться (вместе с сообщением и с открытым ключом обладателя подписи) как данные ввода для процесса верификации, что ведет к принятию простого решения: «подпись действительна/подпись недействительна».

Подписание с восстановлением также создает подпись из сообщения и секретного ключа обладателя подписи. Однако для верификации данной подписи не требуется сообщения в качестве данных ввода. Процесс верификации требует ввода только открытого ключа обладателя подписи, и на выходе процесса выдается сообщение «подпись недействительна» либо (если подпись верна) исходное
сообщение.

Рассмотрим простой пример с механизмом **CKM_RSA_X_509**. Сообщением в данном случае является байтовая строка, которую мы будем считать модулем числа *n* (модуль RSA обладателя подписи). Когда этот механизм используется для обычных электронных подписей (подписей с приложением), подпись вычисляется посредством возведения сообщения в степень, показатель которой равен модулю секретного экспонента обладателя подписи *n*. Чтобы верифицировать эту подпись, верифицирующая сторона возводит подпись в степень, показатель которой равен модулю открытого экспонента обладателя подписи *n*, и принимает подпись как действительную тогда и только тогда, когда результат совпадает с исходным сообщением.

Если **CKM_RSA_X_509** используется для создания подписей с восстановлением, подписи создаются в точности тем же способом. Именно для этого механизма модуль *n любого* числа является действительной подписью. Чтобы восстановить сообщение из подписи, подпись возводится в степень, показатель которой равен модулю *n* открытого экспонента обладателя подписи.

Приложение А
(обязательное)

Объявление констант

В соответствующем файле-заголовке могут находиться следующие определения.

```
#define CK_INVALID_HANDLE     0

#define CKN_SURRENDER         0

#define CK_UNAVAILABLE_INFORMATION     (~0UL)
#define CK_EFFECTIVELY_INFINITE        0

#define CKF_DONT_BLOCK 1

#define CKF_ARRAY_ATTRIBUTE          0x40000000

#define CKU_SO                    0
#define CKU_USER                  1
#define CKU_CONTEXT_SPECIFIC      2

#define CKS_RO_PUBLIC_SESSION     0
#define CKS_RO_USER_FUNCTIONS     1
#define CKS_RW_PUBLIC_SESSION     2
#define CKS_RW_USER_FUNCTIONS     3
#define CKS_RW_SO_FUNCTIONS       4

#define CKO_DATA                  0x00000000
#define CKO_CERTIFICATE           0x00000001
#define CKO_PUBLIC_KEY            0x00000002
#define CKO_PRIVATE_KEY           0x00000003
#define CKO_SECRET_KEY            0x00000004
#define CKO_HW_FEATURE            0x00000005
#define CKO_DOMAIN_PARAMETERS     0x00000006
#define CKO_MECHANISM             0x00000007
#define CKO_VENDOR_DEFINED        0x80000000

#define CKH_MONOTONIC_COUNTER     0x00000001
#define CKH_CLOCK                 0x00000002
#define CKH_USER_INTERFACE        0x00000003
#define CKH_VENDOR_DEFINED        0x80000000

#define CKK_RSA                 0x00000000
#define CKK_DSA                 0x00000001
#define CKK_DH                  0x00000002
#define CKK_ECDSA               0x00000003
#define CKK_EC                  0x00000003
#define CKK_X9_42_DH            0x00000004
#define CKK_KEA                 0x00000005
#define CKK_GENERIC_SECRET      0x00000010
#define CKK_RC2                 0x00000011
#define CKK_RC4                 0x00000012
#define CKK_DES                 0x00000013
#define CKK_DES2                0x00000014
#define CKK_DES3                0x00000015
#define CKK_CAST                0x00000016
#define CKK_CAST3               0x00000017
#define CKK_CAST5               0x00000018
#define CKK_CAST128             0x00000018
#define CKK_RC5                 0x00000019
#define CKK_IDEA                0x0000001A
#define CKK_SKIPJACK            0x0000001B
#define CKK_BATON               0x0000001C
```

```
#define CKK_JUNIPER             0x0000001D
#define CKK_CDMF                        0x0000001E
#define CKK_AES                         0x0000001F
#define CKK_BLOWFISH                    0x00000020
#define CKK_TWOFISH                     0x00000021
#define CKK_VENDOR_DEFINED              0x80000000

#define CKC_X_509                       0x00000000
#define CKC_X_509_ATTR_CERT             0x00000001
#define CKC_WTLS                        0x00000002
#define CKC_VENDOR_DEFINED              0x80000000

#define CKA_CLASS                       0x00000000
#define CKA_TOKEN                       0x00000001
#define CKA_PRIVATE                     0x00000002
#define CKA_LABEL                       0x00000003
#define CKA_APPLICATION                 0x00000010
#define CKA_VALUE                       0x00000011
#define CKA_OBJECT_ID                   0x00000012
#define CKA_CERTIFICATE_TYPE            0x00000080
#define CKA_ISSUER                      0x00000081
#define CKA_SERIAL_NUMBER               0x00000082
#define CKA_AC_ISSUER                   0x00000083
#define CKA_OWNER                       0x00000084
#define CKA_ATTR_TYPES                  0x00000085
#define CKA_TRUSTED                     0x00000086
#define CKA_CERTIFICATE_CATEGORY        0x00000087
#define CKA_JAVA_MIDP_SECURITY_DOMAIN   0x00000088
#define CKA_URL                         0x00000089
#define CKA_HASH_OF_SUBJECT_PUBLIC_KEY  0x0000008A
#define CKA_HASH_OF_ISSUER_PUBLIC_KEY   0x0000008B
#define CKA_CHECK_VALUE                 0x00000090
#define CKA_KEY_TYPE                    0x00000100
#define CKA_SUBJECT                     0x00000101
#define CKA_ID                          0x00000102
#define CKA_SENSITIVE                   0x00000103
#define CKA_ENCRYPT                     0x00000104
#define CKA_DECRYPT                     0x00000105
#define CKA_WRAP                        0x00000106
#define CKA_UNWRAP                      0x00000107
#define CKA_SIGN                        0x00000108
#define CKA_SIGN_RECOVER                0x00000109
#define CKA_VERIFY                      0x0000010A
#define CKA_VERIFY_RECOVER              0x0000010B
#define CKA_DERIVE                      0x0000010C
#define CKA_START_DATE                  0x00000110
#define CKA_END_DATE                    0x00000111
#define CKA_MODULUS                     0x00000120
#define CKA_MODULUS_BITS                0x00000121
#define CKA_PUBLIC_EXPONENT             0x00000122
#define CKA_PRIVATE_EXPONENT            0x00000123
#define CKA_PRIME_1                     0x00000124
#define CKA_PRIME_2                     0x00000125
#define CKA_EXPONENT_1                  0x00000126
#define CKA_EXPONENT_2                  0x00000127
#define CKA_COEFFICIENT                 0x00000128
#define CKA_PRIME                       0x00000130
#define CKA_SUBPRIME                    0x00000131
#define CKA_BASE                        0x00000132
#define CKA_PRIME_BITS                  0x00000133
#define CKA_SUBPRIME_BITS               0x00000134
#define CKA_VALUE_BITS                  0x00000160
#define CKA_VALUE_LEN                   0x00000161
```

```
#define CKA_EXTRACTABLE                 0x00000162
#define CKA_LOCAL                       0x00000163
#define CKA_NEVER_EXTRACTABLE           0x00000164
#define CKA_ALWAYS_SENSITIVE            0x00000165
#define CKA_KEY_GEN_MECHANISM           0x00000166
#define CKA_MODIFIABLE                  0x00000170
#define CKA_ECDSA_PARAMS                0x00000180
#define CKA_EC_PARAMS                   0x00000180
#define CKA_EC_POINT                    0x00000181
#define CKA_SECONDARY_AUTH              0x00000200 /* Deprecated */
#define CKA_AUTH_PIN_FLAGS              0x00000201 /* Deprecated */
#define CKA_ALWAYS_AUTHENTICATE         0x00000202

#define CKA_WRAP_WITH_TRUSTED           0x00000210
#define CKA_WRAP_TEMPLATE         (CKF_ARRAY_ATTRIBUTE|0x00000211)
#define CKA_UNWRAP_TEMPLATE       (CKF_ARRAY_ATTRIBUTE|0x00000212)
#define CKA_HW_FEATURE_TYPE             0x00000300
#define CKA_RESET_ON_INIT               0x00000301
#define CKA_HAS_RESET                   0x00000302
#define CKA_PIXEL_X                     0x00000400
#define CKA_PIXEL_Y                     0x00000401
#define CKA_RESOLUTION                  0x00000402
#define CKA_CHAR_ROWS                   0x00000403
#define CKA_CHAR_COLUMNS                0x00000404
#define CKA_COLOR                       0x00000405
#define CKA_BITS_PER_PIXEL              0x00000406
#define CKA_CHAR_SETS                   0x00000480
#define CKA_ENCODING_METHODS            0x00000481
#define CKA_MIME_TYPES                  0x00000482
#define CKA_MECHANISM_TYPE              0x00000500
#define CKA_REQUIRED_CMS_ATTRIBUTES     0x00000501
#define CKA_DEFAULT_CMS_ATTRIBUTES      0x00000502
#define CKA_SUPPORTED_CMS_ATTRIBUTES    0x00000503
#define CKA_ALLOWED_MECHANISMS    (CKF_ARRAY_ATTRIBUTE|0x00000600)
#define CKA_VENDOR_DEFINED              0x80000000

#define CKM_RSA_PKCS_KEY_PAIR_GEN       0x00000000
#define CKM_RSA_PKCS                    0x00000001
#define CKM_RSA_9796                    0x00000002
#define CKM_RSA_X_509                   0x00000003
#define CKM_MD2_RSA_PKCS                0x00000004
#define CKM_MD5_RSA_PKCS                0x00000005
#define CKM_SHA1_RSA_PKCS               0x00000006
#define CKM_RIPEMD128_RSA_PKCS          0x00000007
#define CKM_RIPEMD160_RSA_PKCS          0x00000008
#define CKM_RSA_PKCS_OAEP               0x00000009
#define CKM_RSA_X9_31_KEY_PAIR_GEN      0x0000000A
#define CKM_RSA_X9_31                   0x0000000B
#define CKM_SHA1_RSA_X9_31              0x0000000C
#define CKM_RSA_PKCS_PSS                0x0000000D
#define CKM_SHA1_RSA_PKCS_PSS           0x0000000E
#define CKM_DSA_KEY_PAIR_GEN            0x00000010
#define CKM_DSA                         0x00000011
#define CKM_DSA_SHA1                    0x00000012
#define CKM_DH_PKCS_KEY_PAIR_GEN        0x00000020
#define CKM_DH_PKCS_DERIVE              0x00000021
#define CKM_X9_42_DH_KEY_PAIR_GEN       0x00000030
#define CKM_X9_42_DH_DERIVE             0x00000031
#define CKM_X9_42_DH_HYBRID_DERIVE      0x00000032
#define CKM_X9_42_MQV_DERIVE            0x00000033
#define CKM_SHA256_RSA_PKCS             0x00000040
#define CKM_SHA384_RSA_PKCS             0x00000041
#define CKM_SHA512_RSA_PKCS             0x00000042
```

```
#define CKM_SHA256_RSA_PKCS_PSS          0x00000043
#define CKM_SHA384_RSA_PKCS_PSS          0x00000044
#define CKM_SHA512_RSA_PKCS_PSS          0x00000045
#define CKM_RC2_KEY_GEN                  0x00000100
#define CKM_RC2_ECB                      0x00000101
#define CKM_RC2_CBC                      0x00000102
#define CKM_RC2_MAC                      0x00000103
#define CKM_RC2_MAC_GENERAL              0x00000104
#define CKM_RC2_CBC_PAD                  0x00000105
#define CKM_RC4_KEY_GEN                  0x00000110
#define CKM_RC4                          0x00000111
#define CKM_DES_KEY_GEN                  0x00000120
#define CKM_DES_ECB                      0x00000121
#define CKM_DES_CBC                      0x00000122
#define CKM_DES_MAC                      0x00000123
#define CKM_DES_MAC_GENERAL              0x00000124
#define CKM_DES_CBC_PAD                  0x00000125
#define CKM_DES2_KEY_GEN                 0x00000130
#define CKM_DES3_KEY_GEN                 0x00000131
#define CKM_DES3_ECB                     0x00000132
#define CKM_DES3_CBC                     0x00000133
#define CKM_DES3_MAC                     0x00000134
#define CKM_DES3_MAC_GENERAL             0x00000135
#define CKM_DES3_CBC_PAD                 0x00000136
#define CKM_CDMF_KEY_GEN                 0x00000140
#define CKM_CDMF_ECB                     0x00000141
#define CKM_CDMF_CBC                     0x00000142
#define CKM_CDMF_MAC                     0x00000143
#define CKM_CDMF_MAC_GENERAL             0x00000144
#define CKM_CDMF_CBC_PAD                 0x00000145
#define CKM_DES_OFB64                    0x00000150
#define CKM_DES_OFB8                     0x00000151
#define CKM_DES_CFB64                    0x00000152
#define CKM_DES_CFB8                     0x00000153
#define CKM_MD2                          0x00000200
#define CKM_MD2_HMAC                     0x00000201
#define CKM_MD2_HMAC_GENERAL             0x00000202
#define CKM_MD5                          0x00000210
#define CKM_MD5_HMAC                     0x00000211
#define CKM_MD5_HMAC_GENERAL             0x00000212
#define CKM_SHA_1                        0x00000220
#define CKM_SHA_1_HMAC                   0x00000221
#define CKM_SHA_1_HMAC_GENERAL           0x00000222
#define CKM_RIPEMD128                    0x00000230
#define CKM_RIPEMD128_HMAC               0x00000231
#define CKM_RIPEMD128_HMAC_GENERAL       0x00000232
#define CKM_RIPEMD160                    0x00000240
#define CKM_RIPEMD160_HMAC               0x00000241
#define CKM_RIPEMD160_HMAC_GENERAL       0x00000242
#define CKM_SHA256                       0x00000250
#define CKM_SHA256_HMAC                  0x00000251
#define CKM_SHA256_HMAC_GENERAL          0x00000252
#define CKM_SHA384                       0x00000260
#define CKM_SHA384_HMAC                  0x00000261
#define CKM_SHA384_HMAC_GENERAL          0x00000262
#define CKM_SHA512                       0x00000270
#define CKM_SHA512_HMAC                  0x00000271
#define CKM_SHA512_HMAC_GENERAL          0x00000272
#define CKM_CAST_KEY_GEN                 0x00000300
#define CKM_CAST_ECB                     0x00000301
#define CKM_CAST_CBC                     0x00000302
#define CKM_CAST_MAC                     0x00000303
#define CKM_CAST_MAC_GENERAL             0x00000304
```

```
#define CKM_CAST_CBC_PAD                0x00000305
#define CKM_CAST3_KEY_GEN                 0x00000310
#define CKM_CAST3_ECB                     0x00000311
#define CKM_CAST3_CBC                     0x00000312
#define CKM_CAST3_MAC                     0x00000313
#define CKM_CAST3_MAC_GENERAL             0x00000314
#define CKM_CAST3_CBC_PAD                 0x00000315
#define CKM_CAST5_KEY_GEN                 0x00000320
#define CKM_CAST128_KEY_GEN               0x00000320
#define CKM_CAST5_ECB                     0x00000321
#define CKM_CAST128_ECB                   0x00000321
#define CKM_CAST5_CBC                     0x00000322
#define CKM_CAST128_CBC                   0x00000322
#define CKM_CAST5_MAC                     0x00000323
#define CKM_CAST128_MAC                   0x00000323
#define CKM_CAST5_MAC_GENERAL             0x00000324
#define CKM_CAST128_MAC_GENERAL           0x00000324
#define CKM_CAST5_CBC_PAD                 0x00000325
#define CKM_CAST128_CBC_PAD               0x00000325
#define CKM_RC5_KEY_GEN                   0x00000330
#define CKM_RC5_ECB                       0x00000331
#define CKM_RC5_CBC                       0x00000332
#define CKM_RC5_MAC                       0x00000333
#define CKM_RC5_MAC_GENERAL               0x00000334
#define CKM_RC5_CBC_PAD                   x00000335
#define CKM_IDEA_KEY_GEN                  0x00000340
#define CKM_IDEA_ECB                      0x00000341
#define CKM_IDEA_CBC                      0x00000342
#define CKM_IDEA_MAC                      0x00000343
#define CKM_IDEA_MAC_GENERAL              0x00000344
#define CKM_IDEA_CBC_PAD                  0x00000345
#define CKM_GENERIC_SECRET_KEY_GEN        0x00000350
#define CKM_CONCATENATE_BASE_AND_KEY      0x00000360
#define CKM_CONCATENATE_BASE_AND_DATA     0x00000362
#define CKM_CONCATENATE_DATA_AND_BASE     0x00000363
#define CKM_XOR_BASE_AND_DATA             0x00000364
#define CKM_EXTRACT_KEY_FROM_KEY          0x00000365
#define CKM_SSL3_PRE_MASTER_KEY_GEN       0x00000370
#define CKM_SSL3_MASTER_KEY_DERIVE        0x00000371
#define CKM_SSL3_KEY_AND_MAC_DERIVE       0x00000372
#define CKM_SSL3_MASTER_KEY_DERIVE_DH     0x00000373
#define CKM_TLS_PRE_MASTER_KEY_GEN        0x00000374
#define CKM_TLS_MASTER_KEY_DERIVE         0x00000375
#define CKM_TLS_KEY_AND_MAC_DERIVE        0x00000376
#define CKM_TLS_MASTER_KEY_DERIVE_DH      0x00000377
#define CKM_TLS_PRF                       0x00000378
#define CKM_SSL3_MD5_MAC                  0x00000380
#define CKM_SSL3_SHA1_MAC                 0x00000381
#define CKM_MD5_KEY_DERIVATION            0x00000390
#define CKM_MD2_KEY_DERIVATION            0x00000391
#define CKM_SHA1_KEY_DERIVATION           0x00000392
#define CKM_SHA256_KEY_DERIVATION         0x00000393
#define CKM_SHA384_KEY_DERIVATION         0x00000394
#define CKM_SHA512_KEY_DERIVATION         0x00000395
#define CKM_PBE_MD2_DES_CBC               0x000003A0
#define CKM_PBE_MD5_DES_CBC               0x000003A1
#define CKM_PBE_MD5_CAST_CBC              0x000003A2
#define CKM_PBE_MD5_CAST3_CBC             0x000003A3
#define CKM_PBE_MD5_CAST5_CBC             0x000003A4
#define CKM_PBE_MD5_CAST128_CBC           0x000003A4
#define CKM_PBE_SHA1_CAST5_CBC            0x000003A5
#define CKM_PBE_SHA1_CAST128_CBC          0x000003A5
#define CKM_PBE_SHA1_RC4_128              0x000003A6
```

```
#define CKM_PBE_SHA1_RC4_40                 0x000003A7
#define CKM_PBE_SHA1_DES3_EDE_CBC             0x000003A8
#define CKM_PBE_SHA1_DES2_EDE_CBC             0x000003A9
#define CKM_PBE_SHA1_RC2_128_CBC              0x000003AA
#define CKM_PBE_SHA1_RC2_40_CBC               0x000003AB
#define CKM_PKCS5_PBKD2                       0x000003B0
#define CKM_PBA_SHA1_WITH_SHA1_HMAC           0x000003C0
#define CKM_WTLS_PRE_MASTER_KEY_GEN           0x000003D0
#define CKM_WTLS_MASTER_KEY_DERIVE            0x000003D1
#define CKM_WTLS_MASTER_KEY_DERVIE_DH_ECC     0x000003D2
#define CKM_WTLS_PRF                          0x000003D3
#define CKM_WTLS_SERVER_KEY_AND_MAC_DERIVE    0x000003D4
#define CKM_WTLS_CLIENT_KEY_AND_MAC_DERIVE    0x000003D5
#define CKM_KEY_WRAP_LYNKS                    0x00000400
#define CKM_KEY_WRAP_SET_OAEP                 0x00000401
#define CKM_CMS_SIG                           0x00000500
#define CKM_SKIPJACK_KEY_GEN                  0x00001000
#define CKM_SKIPJACK_ECB64                    0x00001001
#define CKM_SKIPJACK_CBC64                    0x00001002
#define CKM_SKIPJACK_OFB64                    0x00001003
#define CKM_SKIPJACK_CFB64                    0x00001004
#define CKM_SKIPJACK_CFB32                    0x00001005
#define CKM_SKIPJACK_CFB16                    0x00001006
#define CKM_SKIPJACK_CFB8                     0x00001007
#define CKM_SKIPJACK_WRAP                     0x00001008
#define CKM_SKIPJACK_PRIVATE_WRAP             0x00001009
#define CKM_SKIPJACK_RELAYX                   0x0000100a
#define CKM_KEA_KEY_PAIR_GEN                  0x00001010
#define CKM_KEA_KEY_DERIVE                    0x00001011
#define CKM_FORTEZZA_TIMESTAMP                0x00001020
#define CKM_BATON_KEY_GEN                     0x00001030
#define CKM_BATON_ECB128                      0x00001031
#define CKM_BATON_ECB96                       0x00001032
#define CKM_BATON_CBC128                      0x00001033
#define CKM_BATON_COUNTER                     0x00001034
#define CKM_BATON_SHUFFLE                     0x00001035
#define CKM_BATON_WRAP                        0x00001036
#define CKM_ECDSA_KEY_PAIR_GEN                0x00001040
#define CKM_EC_KEY_PAIR_GEN                   0x00001040
#define CKM_ECDSA                             0x00001041
#define CKM_ECDSA_SHA1                        0x00001042
#define CKM_ECDH1_DERIVE                      0x00001050
#define CKM_ECDH1_COFACTOR_DERIVE             0x00001051
#define CKM_ECMQV_DERIVE                      0x00001052
#define CKM_JUNIPER_KEY_GEN                   0x00001060
#define CKM_JUNIPER_ECB128                    0x00001061
#define CKM_JUNIPER_CBC128                    0x00001062
#define CKM_JUNIPER_COUNTER                   0x00001063
#define CKM_JUNIPER_SHUFFLE                   0x00001064
#define CKM_JUNIPER_WRAP                      0x00001065
#define CKM_FASTHASH                          0x00001070
#define CKM_AES_KEY_GEN                       0x00001080
#define CKM_AES_ECB                           0x00001081
#define CKM_AES_CBC                           0x00001082
#define CKM_AES_MAC                           0x00001083
#define CKM_AES_MAC_GENERAL                   0x00001084
#define CKM_AES_CBC_PAD                       0x00001085
#define CKM_BLOWFISH_KEY_GEN                  0x00001090
#define CKM_BLOWFISH_CBC                      0x00001091
#define CKM_TWOFISH_KEY_GEN                   0x00001092
#define CKM_TWOFISH_CBC                       0x00001093
#define CKM_DES_ECB_ENCRYPT_DATA              0x00001100
#define CKM_DES_CBC_ENCRYPT_DATA              0x00001101
```

```
#define CKM_DES3_ECB_ENCRYPT_DATA              0x00001102
#define CKM_DES3_CBC_ENCRYPT_DATA              0x00001103
#define CKM_AES_ECB_ENCRYPT_DATA               0x00001104
#define CKM_AES_CBC_ENCRYPT_DATA               0x00001105
#define CKM_DSA_PARAMETER_GEN                  0x00002000
#define CKM_DH_PKCS_PARAMETER_GEN              0x00002001
#define CKM_X9_42_DH_PARAMETER_GEN             0x00002002
#define CKM_VENDOR_DEFINED                     0x80000000

#define CKR_OK                                 0x00000000
#define CKR_CANCEL                             0x00000001
#define CKR_HOST_MEMORY                        0x00000002
#define CKR_SLOT_ID_INVALID                    0x00000003
#define CKR_GENERAL_ERROR                      0x00000005
#define CKR_FUNCTION_FAILED                    0x00000006
#define CKR_ARGUMENTS_BAD                      0x00000007
#define CKR_NO_EVENT                           0x00000008
#define CKR_NEED_TO_CREATE_THREADS             0x00000009
#define CKR_CANT_LOCK                          0x0000000A
#define CKR_ATTRIBUTE_READ_ONLY                0x00000010
#define CKR_ATTRIBUTE_SENSITIVE                0x00000011
#define CKR_ATTRIBUTE_TYPE_INVALID             0x00000012
#define CKR_ATTRIBUTE_VALUE_INVALID            0x00000013
#define CKR_DATA_INVALID                       0x00000020
#define CKR_DATA_LEN_RANGE                     0x00000021
#define CKR_DEVICE_ERROR                       0x00000030
#define CKR_DEVICE_MEMORY                      0x00000031
#define CKR_DEVICE_REMOVED                     0x00000032
#define CKR_ENCRYPTED_DATA_INVALID             0x00000040
#define CKR_ENCRYPTED_DATA_LEN_RANGE           0x00000041
#define CKR_FUNCTION_CANCELED                  0x00000050
#define CKR_FUNCTION_NOT_PARALLEL              0x00000051
#define CKR_FUNCTION_NOT_SUPPORTED             0x00000054
#define CKR_KEY_HANDLE_INVALID                 0x00000060
#define CKR_KEY_SIZE_RANGE                     0x00000062
#define CKR_KEY_TYPE_INCONSISTENT              0x00000063
#define CKR_KEY_NOT_NEEDED                     0x00000064
#define CKR_KEY_CHANGED                        0x00000065
#define CKR_KEY_NEEDED                         0x00000066
#define CKR_KEY_INDIGESTIBLE                   0x00000067
#define CKR_KEY_FUNCTION_NOT_PERMITTED         0x00000068
#define CKR_KEY_NOT_WRAPPABLE                  0x00000069
#define CKR_KEY_UNEXTRACTABLE                  0x0000006A
#define CKR_MECHANISM_INVALID                  0x00000070
#define CKR_MECHANISM_PARAM_INVALID            0x00000071
#define CKR_OBJECT_HANDLE_INVALID              0x00000082
#define CKR_OPERATION_ACTIVE                   0x00000090
#define CKR_OPERATION_NOT_INITIALIZED          0x00000091
#define CKR_PIN_INCORRECT                      0x000000A0
#define CKR_PIN_INVALID                        0x000000A1
#define CKR_PIN_LEN_RANGE                      0x000000A2
#define CKR_PIN_EXPIRED                        0x000000A3
#define CKR_PIN_LOCKED                         0x000000A4
#define CKR_SESSION_CLOSED                     0x000000B0
#define CKR_SESSION_COUNT                      0x000000B1
#define CKR_SESSION_HANDLE_INVALID             0x000000B3
#define CKR_SESSION_PARALLEL_NOT_SUPPORTED     0x000000B4
#define CKR_SESSION_READ_ONLY                  0x000000B5
#define CKR_SESSION_EXISTS                     0x000000B6
#define CKR_SESSION_READ_ONLY_EXISTS           0x000000B7
#define CKR_SESSION_READ_WRITE_SO_EXISTS       0x000000B8
#define CKR_SIGNATURE_INVALID                  0x000000C0
#define CKR_SIGNATURE_LEN_RANGE                0x000000C1
```

```
#define CKR_TEMPLATE_INCOMPLETE                0x000000D0
#define CKR_TEMPLATE_INCONSISTENT                 0x000000D1
#define CKR_TOKEN_NOT_PRESENT                     0x000000E0
#define CKR_TOKEN_NOT_RECOGNIZED                  0x000000E1
#define CKR_TOKEN_WRITE_PROTECTED                 0x000000E2
#define CKR_UNWRAPPING_KEY_HANDLE_INVALID         0x000000F0
#define CKR_UNWRAPPING_KEY_SIZE_RANGE             0x000000F1
#define CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT      0x000000F2
#define CKR_USER_ALREADY_LOGGED_IN                0x00000100
#define CKR_USER_NOT_LOGGED_IN                    0x00000101
#define CKR_USER_PIN_NOT_INITIALIZED              0x00000102
#define CKR_USER_TYPE_INVALID                     0x00000103
#define CKR_USER_ANOTHER_ALREADY_LOGGED_IN        0x00000104
#define CKR_USER_TOO_MANY_TYPES                   0x00000105
#define CKR_WRAPPED_KEY_INVALID                   0x00000110
#define CKR_WRAPPED_KEY_LEN_RANGE                 0x00000112
#define CKR_WRAPPING_KEY_HANDLE_INVALID           0x00000113
#define CKR_WRAPPING_KEY_SIZE_RANGE               0x00000114
#define CKR_WRAPPING_KEY_TYPE_INCONSISTENT        0x00000115
#define CKR_RANDOM_SEED_NOT_SUPPORTED             0x00000120
#define CKR_RANDOM_NO_RNG                         0x00000121
#define CKR_DOMAIN_PARAMS_INVALID                 0x00000130
#define CKR_BUFFER_TOO_SMALL                      0x00000150
#define CKR_SAVED_STATE_INVALID                   0x00000160
#define CKR_INFORMATION_SENSITIVE                 0x00000170
#define CKR_STATE_UNSAVEABLE                      0x00000180
#define CKR_CRYPTOKI_NOT_INITIALIZED              0x00000190
#define CKR_CRYPTOKI_ALREADY_INITIALIZED          0x00000191
#define CKR_MUTEX_BAD                             0x000001A0
#define CKR_MUTEX_NOT_LOCKED                      0x000001A1
#define CKR_FUNCTION_REJECTED                     0x00000200
#define CKR_VENDOR_DEFINED                        0x80000000
```

Приложение В
(обязательное)

Профили носителя

Данное приложение описывает «профили», т. е. наборы механизмов, которые носитель должен поддерживать для различных общих типов приложений. Ожидается, что эти наборы будут стандартизованы как части различных приложений, например, в рамках списка требований к модулю, который предоставляет приложению криптографические услуги (в некоторых случаях это может быть носитель Cryptoki). Таким образом, эти профили предназначены для обращения только в данном случае, они не являются частью настоящего стандарта.

В таблице В.1 прведены механизмы, значимые для двух общих типов приложений.

Таблица В.1 – Механизмы и профили

Механизм	Приложение	
	Только аутентификация для правительства	Данные сотовых цифровых пакетов
CKM_DSA_KEY_PAIR_GEN	✓	
CKM_DSA	✓	
CKM_DH_PKCS_KEY_PAIR_GEN		✓
CKM_DH_PKCS_DERIVE		✓
CKM_RC4_KEY_GEN		✓
CKM_RC4		✓
CKM_SHA_1	✓	

В.1 Только аутентификация для правительства

Правительство США стандартизировало алгоритм цифровой подписи DSA (Digital Signature Algorithm) в FIPS PUB 186-2 и алгоритм защищенного хэширования SHA (Secure Hash Algorithm) в FIPS PUB 180-2 для получения списка сообщения. Значимые механизмы включают следующее:

Генерация ключа DSA (512-1024 бита)
DSA (512-1024 бита)
SHA-1

В.2 Данные сотовых цифровых пакетов

Протокол данных сотовых сетевых пакетов (Cellular Digital Packet Data, CDPD) представляет собой набор протоколов для беспроводной связи. Основной набор механизмов, поддерживающих приложения CDPD, включает следующее:

Генерация ключа Диффи – Хеллмана (256-1024 бита)
Деривация ключа Диффи – Хеллмана (256-1024 бита)
Генерация ключа RC4 (40-128 бит)
RC4 (40-128 бит)

(Первоначальное описание параметров безопасности CDPD ограничивает размер ключа Диффи – Хеллмана 256 битами, однако было рекомендовано, чтобы размер был увеличен как минимум до 512 бит.)

В.3 Другие профили

Информируем также читателя о наличии других профилей PKCS #11 v2. – см. [PKCS #11-C] и [PKCS #11-P].

Приложение С
(справочное)

Сравнение Cryptoki с другими API

В настоящем приложении сравнивается Cryptoki с другими криптографическими интерфейсами (API):

- ANSI N13-94 – Guideline X9.TG-12-199X, Using Tessera in Financial Systems: An Application Programming Interface, April 29, 1994
- X/Open GCS-API – Generic Cryptographic Service API, Draft 2, February 14, 1995

C.1 FORTEZZA CIPG, Rev. 1.52

В этом документе описывается интерфейс криптокарты FORTEZZA PCMCIA. Это уровень, подобный Cryptoki. В таблице С.1 перечислены функции FORTEZZA CIPG вместе с эквивалентными им функциями Cryptoki.

Таблица С.1 – FORTEZZA CIPG в сравнении с Cryptoki

FORTEZZA CIPG	Эквивалент Cryptoki
CI_ChangePIN	C_InitPIN, C_SetPIN
CI_CheckPIN	C_Login
CI_Close	C_CloseSession
CI_Decrypt	C_DecryptInit, C_Decrypt, C_DecryptUpdate, C_DecryptFinal
CI_DeleteCertificate	C_DestroyObject
CI_DeleteKey	C_DestroyObject
CI_Encrypt	C_EncryptInit, C_Encrypt, C_EncryptUpdate, C_EncryptFinal
CI_ExtractX	C_WrapKey
CI_GenerateIV	C_GenerateRandom
CI_GenerateMEK	C_GenerateKey
CI_GenerateRa	C_GenerateRandom
CI_GenerateRandom	C_GenerateRandom
CI_GenerateTEK	C_GenerateKey
CI_GenerateX	C_GenerateKeyPair
CI_GetCertificate	C_FindObjects
CI_Configuration	C_GetTokenInfo
CI_GetHash	C_DigestInit, C_Digest, C_DigestUpdate и C_DigestFinal
CI_GetIV	Нет эквивалента
CI_GetPersonalityList	C_FindObjects
CI_GetState	C_GetSessionInfo
CI_GetStatus	C_GetTokenInfo
CI_GetTime	C_GetTokenInfo или C_GetAttributeValue(clock object) ***[предпочтительно]***
CI_Hash	C_DigestInit, C_Digest, C_DigestUpdate и C_DigestFinal
CI_Initialize	C_Initialize
CI_InitializeHash	C_DigestInit
CI_InstallX	C_UnwrapKey
CI_LoadCertificate	C_CreateObject
CI_LoadDSAParameters	C_CreateObject
CI_LoadInitValues	C_SeedRandom
CI_LoadIV	C_EncryptInit, C_DecryptInit
CI_LoadK	C_SignInit
CI_LoadPublicKeyParameters	C_CreateObject
CI_LoadPIN	C_SetPIN

Окончание таблицы С.1

FORTEZZA CIPG	Эквивалент Cryptoki
CI_LoadX	C_CreateObject
CI_Lock	Встроено в управление сеансом
CI_Open	C_OpenSession
CI_RelayX	C_WrapKey
CI_Reset	C_CloseAllSessions
CI_Restore	Встроено в управление сеансом
CI_Save	Встроено в управление сеансом
CI_Select	C_OpenSession
CI_SetConfiguration	Нет эквивалента
CI_SetKey	C_EncryptInit, C_DecryptInit
CI_SetMode	C_EncryptInit, C_DecryptInit
CI_SetPersonality	C_CreateObject
CI_SetTime	Нет эквивалента
CI_Sign	C_SignInit, C_Sign
CI_Terminate	C_CloseAllSessions
CI_Timestamp	C_SignInit, C_Sign
CI_Unlock	Встроено в управление сеансом
CI_UnwrapKey	C_UnwrapKey
CI_VerifySignature	C_VerifyInit, C_Verify
CI_VerifyTimestamp	C_VerifyInit, C_Verify
CI_WrapKey	C_WrapKey
CI_Zeroize	C_InitToken

С.2 GCS-API

Представленный стандарт относит API к службам обеспечения безопасности высокого уровня, таким как аутентификация идентичности (подтверждение подлинности) и подтверждение происхождения данных, обеспечение безотказности, разделение и защита. Этот уровень более высок, чем уровень Cryptoki. В таблице С.2 перечислены функции GCS-API вместе с функциями Cryptoki, применяемыми для осуществления данных функций. Следует отметить, что полная поддержка GCS-API предусмотрена для последующих версий Cryptoki.

Таблица С.2 – GCS-API в сравнении с Cryptoki

GCS-API	Реализация в Cryptoki
retrieve_CC	
release_CC	
generate_hash	C_DigestInit, C_Digest
generate_random_number	C_GenerateRandom
generate_checkvalue	C_SignInit, C_Sign, C_SignUpdate, C_SignFinal
verify_checkvalue	C_VerifyInit, C_Verify, C_VerifyUpdate, C_VerifyFinal
data_encipher	C_EncryptInit, C_Encrypt, C_EncryptUpdate, C_EncryptFinal
data_decipher	C_DecryptInit, C_Decrypt, C_DecryptUpdate, C_DecryptFinal
create_CC	
derive_key	C_DeriveKey
generate_key	C_GenerateKey
store_CC	
delete_CC	
replicate_CC	
export_key	C_WrapKey

Окончание таблицы С.2

GCS-API	Реализация в Cryptoki
import_key	C_UnwrapKey
archive_CC	C_WrapKey
restore_CC	C_UnwrapKey
set_key_state	
generate_key_pattern	
verify_key_pattern	
derive_clear_key	C_DeriveKey
generate_clear_key	C_GenerateKey
load_key_parts	
clear_key_encipher	C_WrapKey
clear_key_decipher	C_UnwrapKey
change_key_context	
load_initial_key	
generate_initial_key	
set_current_master_key	
protect_under_new_master_key	
protect_under_current_master_key	
initialise_random_number_generator	C_SeedRandom
install_algorithm	
de_install_algorithm	
disable_algorithm	
enable_algorithm	
set_defaults	

Приложение D
(обязательное)

Сведения об интеллектуальной собственности

Криптосистема с открытым ключом RSA описана в патенте США U.S. Patent 4,405,829, срок действия которого истек 20 сентября 2000 г. Блочный шифр RC5 защищен патентами США 5,724,428 и 5,835,600. Компания RSA Security Inc. не подавала других патентных заявок на решения, описанные в данном документе, однако права компании могут распространяться на отдельные методики, лежащие в основе описанных систем.

RSA, RC2 и RC4 являются зарегистрированными торговыми марками компании RSA Security Inc. RC5 является торговой маркой компании RSA Security Inc.

CAST, CAST3, CAST5 и CAST128 являются зарегистрировнными торговыми марками компании Entrust Technologies. OS/2 и CDMF (Commercial Data Masking Facility) являются зарегистрированными торговыми марками корпорации IBM (International Business Machines Corporation). LYNKS является зарегистрированной торговой маркой SPYRUS Corporation. IDEA является зарегистрированной торговой маркой компании Ascom Systec. Windows, Windows 3.1, Windows 95, Windows NT и Developer Studio являются зарегистрированными торговыми марками Microsoft Corporation. UNIX является зарегистрированной торговой маркой компании UNIX System Laboratories. FORTEZZA является зарегистрированной торговой маркой Агентства национальной безопасности США (National Security Agency, NSA).

Лицензия на копирование данного документа предоставляется при условии, что во всех материалах, упоминающих данный документ либо содержащих ссылки на него, он будет обозначен как «RSA Security Inc. Public-Key Cryptography Standards (PKCS)» – «Стандарты криптографии с открытым ключом компании RSA Security Inc.».

Компания RSA Security Inc. не предоставляет никаких иных сведений в отношении заявок на интеллектуальную собственность от других сторон. Эти обязанности вменяются в ответственность пользователю.

Приложение Е
(справочное)

Метод работы носителя с несколькими пользовательскими профилями посредством Cryptoki (исключен из числа рекомендованных)

Примечание – Поддержка данного режима работы может быть представлена для обратной совместимости (подробнее см. PKCS11 V 2.11).

Приложение F
(справочное)

История переработок

Настоящий стандарт представляет собой первоначальную версию PKCS #11 v2.20.

Ответственный за выпуск *В. Л. Гуревич*

Сдано в набор 05.05.2009. Подписано в печать 12.06.2009. Формат бумаги 60×84/8. Бумага офсетная.
Гарнитура Arial. Печать ризографическая. Усл. печ. л. 30,45 Уч.- изд. л. 18,84 Тираж 30 экз. Заказ 369

Издатель и полиграфическое исполнение:
Научно-производственное республиканское унитарное предприятие
«Белорусский государственный институт стандартизации и сертификации» (БелГИСС)
ЛИ № 02330/0549409 от 08.04.2009.
ул. Мележа, 3, 220113, Минск.