Расширение PKCS#11 для использования стандартов ГОСТ 34.12-2018, ГОСТ 34.13-2018, ГОСТ 34.10-2018, ГОСТ 34.11-2018

Москва

2024

Содержание

Аннотация.

Данный документ определяет расширение спецификаций программного интерфейса PKCS#11, версии 3.0 и выше, для использования криптографических алгоритмов ГОСТ 34.12-2018, ГОСТ 34.13-2018, ГОСТ 34.10-2018, ГОСТ 34.11-20182, а также криптографических механизмов, построенных на их основе. Документ подготовлен на основе методических рекомендаций ТК 26 МР 26.2.007-2017 «Расширение PKCS#11 для использования российских стандартов ГОСТ Р 34.10-2012 и ГОСТ Р 34-11-2012» и проекта методических рекомендаций ТК 26 «Расширение PKCS#11 для использования стандартов ГОСТ 34.12-2018 и ГОСТ 34.13-2018».

1. Область применения.

Настоящие рекомендации предназначены для применения в информационных системах, использующих российские криптографические алгоритмы и стандарт PKCS#11. Стандарт PKCS#11 определяет программный интерфейс доступа к криптографическим устройствам. Этот документ расширяет PKCS#11 и определяет интерфейс вызова функций с механизмами, построенными на основе российских криптографических алгоритмов. Интерфейс может использоваться как при построении реализации механизмов в нотации PKCS#11, так и при разработке СКЗИ, построенных с использованием этих реализаций.

2. Список ссылок.

В настоящем документе использованы ссылки на следующие стандарты и рекомендации:

- **ГОСТ 34.10-2018** — «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи», Межгосударственный стандарт, ГОСТ 34.10-2018, , Межгосударственный совет по стандартизации, метрологии и сертификации, Стандартинформ, 2018.
- **ГОСТ 34.11-2018** — «Информационная технология. Криптографическая защита информации. Функция хэширования», Межгосударственный стандарт, ГОСТ 34.11-2018, Межгосударственный совет по стандартизации, метрологии и сертификации, Стандартинформ, 2018.
- **ГОСТ 34.12-2018 —** «Информационная технология. Криптографическая защита информации. Блочные шифры», Межгосударственный стандарт, ГОСТ 34.12-2018, Межгосударственный совет по стандартизации, метрологии и сертификации, Стандартинформ, 2018.
- **ГОСТ 34.13-2018 —** «Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров (с Поправкой, с Изменением №1)», Межгосударственный стандарт, ГОСТ 34.13-2018, Межгосударственный совет по стандартизации, метрологии и сертификации, Стандартинформ, 2019.
- **ТК26АЛГ** — Р 50.1.113-2016. «Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов электронной цифровой подписи и функции хэширования». Рекомендации по стандартизации. Стандартинформ, 2016.
- **ТК26АЛГ2** — Р 1323565.1.017-2018 «Информационная технология. Криптографическая защита информации. Криптографические алгоритмы, сопутствующие применению алгоритмов блочного шифрования». Рекомендации по стандартизации. Стандартинформ, 2018.
- **ТК26TLS_1.2** — Р 1323565.1.020-2020 «Информационная технология. Криптографическая защита информации. Использование российских криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.2)». Рекомендации по стандартизации. Стандартинформ, 2021.
- **ТК26TLS_1.3** — Р 1323565.1.030-2020 «Информационная технология. Криптографическая защита информации. Использование российских криптографических алгоритмов в протоколе безопасности транспортного уровня (TLS 1.3)». Рекомендации по стандартизации. Стандартинформ, 2020.

- **PKCS#11_Base —** PKCS #11. Cryptographic Token Interface. Base Specification. Version 3.0
- **PKCS#11_Mech —** PKCS #11. Cryptographic Token Interface. Current Mechanisms Specification. Version 3.0
- **RFC2104 —** HMAC: Keyed-Hashing for Message Authentication, H. Krawczyk, M. Bellare, R. Canetti, February 1997.
- **RFC7836** — Guidelines on the Cryptographic Algorithms to Accompany the Usage of Standards GOST R 34.10-2012 and GOST R 34.11-2012, S. Smyshlyaev, E. Alekseev, I. Oshkin, V. Popov, S. Leontiev, V. Podobaev, D. Belyavsky, March 2016.

3. Замечания.

3.1 Об использовании численных идентификаторов.

Во избежании совпадения значений вводимых в данном документе численных идентификаторов с базовыми стандартами и другими возможными расширениями значения выбираются в соответствии со следующими правилами:

1) признаком нестандартного значения (определяемого производителем) является взведенный старший бит (0x80000000) численного идентификатора;

2) в каждом из самостоятельных «пространств имен» определений значения выбираются произвольно с учетом уникальной базы и идентификатора производителя.

В качестве идентификатора производителя выбрано следующее значение:

```
#define NSSCK_VENDOR_PKCS11_RU_TEAM 0xd4321000 //0x80000000|0x54321000
#define CK_VENDOR_PKCS11_RU_TEAM_TC26 NSSCK_VENDOR_PKCS11_RU_TEAM
```

Заголовочный файл со списком используемых идентификаторов приведен в обязательном приложении 1.

3.2 О транслитерации названия "Кузнечик".

В **[ГОСТ 34.12-2018]** блочный шифр "Кузнечик" транслитерируется как "Kuznechik". В RFC 7801 используется транслитерация "Kuznyechik".

Так как данный документ основан именно на стандарте **[ГОСТ 34.12-2018]**, основным способом написания является именно "Kuznechik". Тем не менее написание "Kuznyechik" допускается.

4. Расширение PKCS#11 для использования стандартов ГОСТ 34.12-2018, ГОСТ 34.13-2018

4.1 Типы ключа алгоритма ГОСТ 34.12-2018.

Для использования криптографических алгоритмов **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** вводятся типы ключей:

```
CKK_MAGMA
CKK_KUZNECHIK
```

Значения `CKK_MAGMA` и `CKK_KUZNECHIK` могут являться значением атрибута `CKA_KEY_TYPE` для объектов типа `CKO_SECRET_KEY`.

Ключ со значением `CKK_MAGMA` в атрибуте `CKA_KEY_TYPE` (далее ключ алгоритма Магма) предназначен для использования в блочном шифре «Магма» («Magma») с длиной секретного ключа 256 бит и с длиной блока 64 бит по стандарту **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]**.

Ключ со значением `CKK_KUZNECHIK` в атрибуте `CKA_KEY_TYPE` (далее ключ алгоритма Кузнечик) предназначен для использования в блочном шифре «Кузнечик» («Kuznechik») с длиной секретного ключа 256 бит и с длиной блока 128 бит по стандарту **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]**.

Для использования алгоритмов KExp15 экспорта и импорта ключей в соответствии с документом **[ТК26АЛГ2, раздел 5]** вводятся типы ключей:

```
CKK_MAGMA_TWIN_KEY
CKK_KUZNECHIK_TWIN_KEY
```

Значения `CKK_MAGMA_TWIN_KEY` и `CKK_KUZNECHIK_TWIN_KEY` могут являться значением атрибута `CKA_KEY_TYPE` для `CKO_SECRET_KEY`.

Ключ со значением `CKK_MAGMA_TWIN_KEY` в атрибуте `CKA_KEY_TYPE` предназначен для использования в алгоритме экспорта и импорта ключей, построенном на основании блочного шифра «Магма» («Magma») с длиной секретного ключа 256 бит и с длиной блока 64 бит по стандарту **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]**.

Ключ со значением `CKK_KUZNECHIK_TWIN_KEY` в атрибуте `CKA_KEY_TYPE` предназначен для использования в алгоритме экспорта и импорта ключей, построенном на основании блочного шифра «Кузнечик» («Kuznechik») с длиной секретного ключа 256 бит и с длиной блока 128 бит по стандарту **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]**.

Ключи с типами CKK_MAGMA_TWIN_KEY и CKK_KUZNECHIK_TWIN_KEY по сути являются объединениями двух соответствующих ключей: ключа вычисления MAC и ключа шифрования.

Если у извлекаемого ключа запросить значение CKA_VALUE, полученное значение размера 512 бит интерпретируется как последовательно записанные ключ вычисления MAC и ключ шифрования в виде байтовых массивов.

4.2 Механизм генерации ключей алгоритма Кузнечик.

Для генерации ключей по стандарту **[ГОСТ 34.12-2018]** с типом CKK_KUZNECHIK предназначенных для использования в блочном шифре «Кузнечик» вводится механизм

CKM_KUZNECHIK_KEY_GEN

Этот механизм используется в функции C_GenerateKey.

Механизм не использует параметров. Создает ключ длиной 256 бит. CKA_VALUE_LEN 32 байта.

Пример использования механизма CKM_KUZNECHIK_KEY_GEN приведен в приложении 2.1.

4.3 Шифрование алгоритмом Кузнечик в режиме простой замены.

Для шифрования в соответствии со стандартами **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** по алгоритму Кузнечик с длиной блока 128 бит в режиме простой замены вводится механизм

CKM_KUZNECHIK_ECB

Механизм используется в функциях C_EncryptInit и C_DecryptInit для начала шифрования. Они используют для шифрования ключ, у которого в качестве класса объекта указан CKO_SECRET_KEY и в атрибуте CKA_KEY_TYPE стоит значение CKK_KUZNECHIK.

Механизм не требует параметров.

Длина шифруемой информации должна быть кратна длине блока, режим дополнения не используется. Этот механизм может применяться для преобразования ключевой информации.

Пример использования механизма CKM_KUZNECHIK_ECB приведен в приложении 2.2.

4.4 Шифрование алгоритмом Кузнечик в режиме гаммирования.

Для шифрования в соответствии со стандартами **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** по алгоритму Кузнечик с длиной блока 128 бит в режиме гаммирования (CTR) с преобразованием ключаАСРКМ вводится механизм

CKM_KUZNECHIK_CTR_ACPKM

Механизм используется в функциях C_EncryptInit и C_DecryptInit для начала шифрования. Они используют для шифрования ключ, у которого в качестве класса объекта указан CKO_SECRET_KEY и в атрибуте CKA_KEY_TYPE стоит значение CKK_KUZNECHIK**.**

Механизм использует два параметра: период смены ключа (в **[ГОСТ 34.13-2018]** обозначен N) и синхропосылку (вектор инициализации), длина которой равна половине длины блока. Параметры задаются в виде последовательно записанного периода смены ключа 32-битного целого, представленного в big-endian формате, и синхропосылки в виде байтового массива. Если период смены ключа установлен в нуль, ключ по алгоритму ACPKM не меняется и механизм совпадает с режимом CTR, описанном в **[ГОСТ 34.13-2018]**.

Пример использования механизма CKM_KUZNECHIK_CTR_ACPKM приведен в приложении 2.3.

4.5 Вычисление имитовставки для алгоритма Кузнечик.

Для вычисления и проверки имитовставки по алгоритму Кузнечик в соответствии со стандартами **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** с длиной блока 128 бит в режиме выработки имитовставки вводится механизм

CKM_KUZNECHIK_MAC

Механизм используется в функциях C_SignInit и C_VerifyInit для начала процедуры вычисления и проверки MAC. Они используют ключ, у которого в качестве класса объекта указан CKO_SECRET_KEY и в атрибуте CKA_KEY_TYPE стоит значение CKK_KUZNECHIK**.**

Механизм не использует параметров. Размер получаемого значения MAC равен 16 байтам.

Пример использования механизма CKM_KUZNECHIK_MAC приведен в приложении 2.4.

4.6 Экспорт и импорт ключей алгоритма Кузнечик.

В документе **[ТК26АЛГ2, раздел 5]** определен алгоритм экспорта KExp15 и импорта KImp15 ключей.

Для использования этого алгоритма экспорта, построенного на основании алгоритма шифрования **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** по алгоритму Кузнечик с длиной блока 128 бит вводится механизм

`CKM_KUZNECHIK_KEXP_15_WRAP`

Этот механизм используется в функциях `C_Wrap` и `C_UnWrap`.

В качестве ключа, на котором производится экспорт и импорт ключей используется ключ типа `CKK_KUZNECHIK_TWIN_KEY`.

Механизм может применяться к любым ключам.

Механизм использует один параметр - синхропосылку (вектор инициализации) в виде байтового массива, длина которого равна половине длины блока.

Пример использования механизма `CKM_KUZNECHIK_KEXP_15_WRAP` приведен в приложении 2.5.

4.7 Использование режима MGM на основе алгоритма Кузнечик.

Для шифрования в соответствии со стандартами **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** по алгоритму Кузнечик с длиной блока 128 бит в режиме аутентифицированного шифрования с ассоциированными данными (MGM) вводится механизм

`CKM_KUZNECHIK_MGM`

Этот механизм используется в функциях `C_EncryptInit` и `C_DecryptInit`.

В качестве ключа используется ключ типа `CKK_KUZNECHIK`.

Механизм в функции `C_Encrypt` дописывает значение имитовставки в конец зашифрованных данных. При расшифровании функцией `C_Decrypt` механизм проверяет корректность имитовставки перед выдачей расшифрованного текста.

Механизм использует один параметр – структуру типа `CK_GCM_PARAMS`. Поля этой структуры задают следующие параметры механизма:

- `pIv` – байтовый массив, задающий синхропосылку алгоритма, которая в описании алгоритма **[ГОСТ 34.13-2018]** обозначена `IV`.
- `ulIvLen` – байтовая длина поля `pIv`.
- `ulIvBits` – битовая длина поля `pIv`. Стандарт PKCS#11 не рекомендует использовать это поле.

- `pAAD` – байтовый массив, содержащий ассоциированные данные, которые в описании алгоритма **[ГОСТ 34.13-2018]** обозначены `A`.
- `ulAADLen` – длина поля `pAAD`;
- `ulTagBits` – битовая длина MAC, которая в описании алгоритма **[ГОСТ 34.13-2018]** обозначена `s`.

Пример использования механизма `CKM_KUZNECHIK_MGM` приведен в приложении 2.6.

4.8 Механизм генерации ключей алгоритма Магма.

Для генерации ключей по стандарту **[ГОСТ 34.12-2018]** с типом CKK_MAGMA предназначенных для использования в блочном шифре «Магма» вводится механизм

CKM_MAGMA_KEY_GEN

Этот механизм используется в функции C_GenerateKey.

Механизм не использует параметров. Создает ключ длиной 256 бит. CKA_VALUE_LEN 32 байта.

Пример использования механизма CKM_MAGMA_KEY_GEN приведен в приложении 2.7.

4.9 Шифрование алгоритмом Магма в режиме простой замены.

Для шифрования в соответствии со стандартами **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** по алгоритму Магма с длиной блока 64 бит в режиме простой замены вводится механизм

CKM_MAGMA_ECB

Механизм используется в функциях C_EncryptInit и C_DecryptInit для начала шифрования. Они используют для шифрования ключ, у которого в качестве класса объекта указан CKO_SECRET_KEY и в атрибуте CKA_KEY_TYPE стоит значение CKK_MAGMA**.**

Механизм не требует параметров.

Длина шифруемой информации должна быть кратна длине блока, режим дополнения не используется. Этот механизм может применяться для преобразования ключевой информации.

Пример использования механизма CKM_MAGMA_ECB приведен в приложении 2.8.

4.10 Шифрование алгоритмом Магма в режиме гаммирования.

Для шифрования в соответствии со стандартами **[ГОСТ 34.12-2018]** и [**ГОСТ 34.13-2018]** по алгоритму Магма с длиной блока 64 бит в режиме гаммирования CTR с преобразованием ключаACPKM вводитсямеханизм

CKM_MAGMA_CTR_ACPKM

Механизм используется в функциях C_EncryptInit и C_DecryptInit для начала шифрования. Они используют для шифрования ключ, у которого в качестве класса объекта указан CKO_SECRET_KEY и в атрибуте CKA_KEY_TYPE стоит значение CKK_MAGMA**.**

Механизм использует два параметра: период смены ключа (в **[ГОСТ 34.13-2018]** обозначен N) и синхропосылку (вектор инициализации), длина которой равна половине длины блока. Параметры задаются в виде последовательно записанного периода смены ключа 32-битного целого, представленного в big-endian формате, и синхропосылки в виде байтового массива. Если период смены ключа установлен в нуль, ключ по алгоритму ACPKM не меняется и механизм совпадает с режимом CTR, описанном в **[ГОСТ 34.13-2018]**.

Пример использования механизма CKM_MAGMA_CTR_ACPKM приведен в приложении 2.9.

4.11 Вычисление имитовставки для алгоритма Магма.

Для вычисления и проверки имитовставки по алгоритму Магма в соответствии со стандартами **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** с длиной блока 64 бит в режиме выработки имитовставки служит механизм

CKM_MAGMA_MAC

Механизм используется в функциях C_SignInit и C_VerifyInit для начала процедуры вычисления и проверки MAC. Они используют ключ, у которого в качестве класса объекта указан CKO_SECRET_KEY и в атрибуте CKA_KEY_TYPE стоит значение CKK_MAGMA**.**

Механизм не использует параметров. Размер получаемого значения MAC равен 8 байт.

Пример использования механизма CKM_MAGMA_MAC приведен в приложении 2.10.

4.12 Экспорт и импорт ключей алгоритма Магма.

В документе **[ТК26АЛГ2, раздел 5]** определен алгоритм экспорта KExp15 и импорта KImp15 ключей.

Для использования этого алгоритма, построенного на основании алгоритма шифрования **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** по алгоритму Магма с длиной блока 64 бит, вводится механизм

CKM_MAGMA_KEXP_15_WRAP

Этот механизм используется в функциях C_Wrap и C_UnWrap.

В качестве ключа, на котором производится экспорт и импорт ключей используется ключ типа CKK_MAGMA_TWIN_KEY.

Механизм может применяться к любым ключам.

Механизм использует один параметр - синхропосылку (вектор инициализации) в виде байтового массива, длина которого равна половине длины блока.

Пример использования механизма CKM_MAGMA_KEXP_15_WRAP приведен в приложении 2.11.

4.13 Использование режима MGM на основе алгоритма Магма.

Для шифрования в соответствии со стандартами **[ГОСТ 34.12-2018]** и **[ГОСТ 34.13-2018]** по алгоритму Магма с длиной блока 64 бит в режиме аутентифицированного шифрования с ассоциированными данными (MGM) вводится механизм

CKM_MAGMA_MGM

Этот механизм используется в функциях C_EncryptInit и C_DecryptInit.

В качестве ключа используется ключ типа CKK_MAGMA.

Механизм в функции C_Encrypt дописывает значение имитовставки в конец зашифрованных данных. При расшифровании функцией C_Decrypt механизм проверяет корректность имитовставки перед выдачей расшифрованного текста.

Механизм использует один параметр – структуру типа CK_GCM_PARAMS. Поля этой структуры задают следующие параметры механизма:

- pIv – байтовый массив, задающий синхропосылку алгоритма, которая в описании алгоритма **[ГОСТ 34.13-2018]** обозначена IV.
- ulIvLen – байтовая длина поля pIv.
- ulIvBits – битовая длина поля pIv. Стандарт PKCS#11 не рекомендует использовать это поле.
- pAAD – байтовый массив, содержащий ассоциированные данные, которые в описании алгоритма **[ГОСТ 34.13-2018]** обозначены A.
- ulAADLen – длина поля pAAD;

- `ulTagBits` – битовая длина имитовставки MAC, которая в описании алгоритма **[ГОСТ 34.13-2018]** обозначена `s`.

Пример использования механизма CKM_MAGMA_MGM приведен в приложении 2.12.

4.14 Механизм диверсификации ключа CKM_KDF_HMAC3411_2012_256.

В документе **[ТК26АЛГ, раздел 4.4]** определен алгоритм KDF_GOSTR3411_2012_256, который задает функцию диверсификации для порождения ключевого материала длиной 256 бит, построенный на основе алгоритма HMAC_GOSTR3411_2012_256. Для использования этого алгоритма диверсификации ключей вводится механизм

`CKM_KDF_HMAC3411_2012_256`

Используется в функции `C_DeriveKey`.

Механизм может применяться к ключам шифрования `CKK_GOST28147`, `CKK_KUZNECHIK`, `CKK_MAGMA` и `CKK_GENERIC_SECRET`.

Механизм использует один параметр, который представляет собой конкатенацию байтовых строк `0x01|label|0x00|seed|0x01|0x00`, то есть ту строку, которая передается на вход функции HMAC.

К механизму применяются следующие правила об извлекаемости и чувствительности результирующего ключа:

- Атрибуты `CKA_SENSITIVE` и `CKA_EXTRACTABLE` в шаблоне нового ключа могут принимать значения `CK_TRUE` или `CK_FALSE`. Если атрибуты не заданы, то они имеют значения по умолчанию, которые могут зависеть от устройства или других атрибутов (**[PKCS#11_Base раздел 4.10]**).
- Если атрибут `CKA_ALWAYS_SENSITIVE` исходного ключа имеет значение `CK_FALSE`, то соответствующий атрибут результирующего ключа также будет иметь значение `CK_FALSE`. Если атрибут `CKA_ALWAYS_SENSITIVE` исходного ключа имеет значение `CK_TRUE`, то значение соответствующего атрибута результирующего ключа будет совпадать со значением атрибута `CKA_SENSITIVE` результирующего ключа.
- Если атрибут `CKA_NEVER_EXTRACTABLE` исходного ключа имеет значение `CK_FALSE`, то соответствующий атрибут результирующего ключа также будет иметь значение `CK_FALSE`. Если атрибут `CKA_NEVER_EXTRACTABLE` исходного ключа имеет значение `CK_TRUE`, то значение соответствующего атрибута результирующего ключа будет противоположным значению атрибута `CKA_EXTRACTABLE` результирующего ключа.

Пример использования механизма CKM_KDF_HMAC3411_2012_256 приведен в приложении 2.13.

4.15 Использование механизма CKM_CONCATENATE_BASE_AND_KEY для объединения ключей.

Для создания ключей типа CKK_KUZNECHIK_TWIN_KEY и CKK_MAGMA_TWIN_KEY можно использовать стандартный механизм CKM_CONCATENATE_BASE_AND_KEY.

Этим механизмом можно объединить два обычных ключа и получить один объединенный ключ. При объединении двух ключей с типом CKK_KUZNECHIK получается ключ типа CKK_KUZNECHIK_TWIN_KEY. При объединении двух ключей с типом CKK_MAGMA получается ключ типа CKK_MAGMA_TWIN_KEY.

Для получения объединенного ключа необходимо добавить к ключу вычисления MAC ключ шифрования. Для этого требуется передать описатель ключа вычисления MAC в качестве параметра hBaseKey функции C_DeriveKey, а описатель ключа шифрования передать в качестве параметра механизма CKM_CONCATENATE_BASE_AND_KEY.

К механизму применяются стандартные правила об извлекаемости и чувствительности результирующего ключа, определенные в **[PKCS#11_Mech, пункт 2.43.3]** следующим образом:

- Если значения атрибута CKA_SENSITIVE обоих базовых ключей принимают значения CK_TRUE, то атрибут CKA_SENSITIVE результирующего ключа также принимает значение CK_TRUE. В противном случае, используется значение атрибута, заданного в шаблоне создаваемого ключа. Если атрибут не задан, то он имеет значение по умолчанию, которое может зависеть от устройства или других атрибутов (**[PKCS#11_Base раздел 4.10]**).
- Если значения атрибута CKA_EXTRACTABLE обоих базовых ключей принимают значения CK_FALSE, то атрибут CKA_EXTRACTABLE результирующего ключа также принимает значение CK_FALSE. В противном случае, используется значение атрибута, заданного в шаблоне создаваемого ключа. Если атрибут не задан, то он имеет значение по умолчанию, которое может зависеть от устройства или других атрибутов (**[PKCS#11_Base раздел 4.10]**).

- Атрибут CKA_ALWAYS_SENSITIVE результирующего ключа имеет значение CK_TRUE тогда и только тогда, когда соответствующий атрибут обоих базовых ключей также будет иметь значение CK_TRUE.
- Атрибут CKA_NEVER_EXTRACTABLE результирующего ключа имеет значение CK_TRUE тогда и только тогда, когда соответствующий атрибут обоих базовых ключец также будет иметь значение CK_TRUE.

Пример использования механизма CKM_CONCATENATE_BASE_AND_KEY приведен в приложении 2.14.

4.16 Механизм диверсификации CKM_KDF_TREE_GOSTR3411_2012_256.

В документе **[ТК26АЛГ, раздел 4.5]** определен алгоритм KDF_TREE_GOSTR3411_2012_256 на основе хэш-функции, определенной в **[ГОСТ 34.11-2018]**, который задает функцию диверсификации ключей. Для реализации этого алгоритма вводится механизм

`CKM_KDF_TREE_GOSTR3411_2012_256`

Он используется в функции `C_DeriveKey`.

Механизм может применяться к ключам шифрования `CKK_GOST28147`, `CKK_KUZNECHIK`, `CKK_MAGMA` и `CKK_GENERIC_SECRET`.

Результатом работы этого механизма могут быть ключи типа `CKK_GOST28147`, `CKK_KUZNECHIK`, `CKK_MAGMA`, `CKK_GENERIC_SECRET`, `CKK_KUZNECHIK_TWIN_KEY` и `CKK_MAGMA_TWIN_KEY`.

Механизм использует один параметр - структуру типа `CK_KDF_TREE_GOST_PARAMS`. Поля этой структуры задают следующие параметры механизма:

- `pLabel`, `pSeed` – байтовые строки, параметры, задаваемые протоколом, соответствуют величинам `label` и `seed` в описании алгоритма **[ТК26АЛГ, раздел 4.5]**.
- `ulR` – количество байт в байтовом представлении счетчика итераций, с возможными значениями 1, 2, 3, 4, соответствуетвличине `R` в описании алгоритма **[ТК26АЛГ, раздел 4.5]**.
- `ulL` – необходимая байтовая длина вырабатываемого ключевого материала, отличается от обозначения `L` в описании алгоритма **[ТК26АЛГ, раздел 4.5]**, который обозначает битовую длину.
- `ulOffset` определяет байтовое смещение, в последовательности ключевого материала, начиная с которого полученные байты используются для получения диверсифицированного ключа - результата.

Параметры `pLabel`, `pSeed`, `ulL`, `ulR` полностью определяют байтовую последовательность ключевого материала длины `ulL`, которую порождает алгоритм.

К механизму применяются следующие правила об извлекаемости и чувствительности результирующего ключа:

- Атрибуты CKA_SENSITIVE и CKA_EXTRACTABLE в шаблоне нового ключа могут принимать значения CK_TRUE или CK_FALSE. Если атрибуты не заданы, то они имеют значения по умолчанию, которые могут зависеть от устройства или других атрибутов (**[PKCS#11_Base раздел 4.10]**).
- Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет совпадать со значением атрибута CKA_SENSITIVE результирующего ключа.
- Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет противоположным значению атрибута CKA_EXTRACTABLE результирующего ключа.

Пример использования механизма CKM_KDF_TREE_GOSTR3411_2012_256 приведен в приложении 2.15.

5. Расширение PKCS#11 для использования стандартов ГОСТ 34.10-2018, ГОСТ 34.11-2018

5.1 Тип ключа алгоритма ГОСТ 34.10-2018.

Данное расширение PKCS#11 вводит определение в пространстве значений типов ключей:

```
#define CKK_GOSTR3410_512 (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x003)
```

Значение CKK_GOSTR3410_512 может являться значением атрибута CKA_KEY_TYPE для CKO_PUBLIC_KEY и CKO_PRIVATE_KEY.

Ключ с типом CKK_GOSTR3410_512 предназначен для использования с алгоритмом **[ГОСТ 34.10-2018]** с длиной ключа подписи 512 бит.

Такой ключ может использоваться с механизмами

```
CKM_GOSTR3410_512_KEY_PAIR_GEN
CKM_GOSTR3410_512
CKM_GOSTR3410_2012_DERIVE
```

5.2 Наборы параметров.

В качестве параметров алгоритма **[ГОСТ 34.10-2018]** допускается использование параметров эллиптических кривых, описанных в документе **[ТК26ЭК]**:

В объектах ключей и шаблонах, которые передаются в функции, в качестве значений атрибутов `CKA_GOSTR3410_PARAMS`, параметры эллиптических кривых представляются в виде соответствующих им идентификаторов в кодировке ASN.1 в соответствии с **[ГОСТ Р ИСО/МЭК 8824-1]**.

Набор параметров, с идентификатором 1.2.643.7.1.2.1.2.1, является набором по умолчанию для ключа подписи длины 512 бит. Если в параметрах механизма не заданы идентификаторы эллиптических кривых, то используются наборы по умолчанию.

5.3 Использование объектов типа CKO_DOMAIN_PARAMETERS.

Реализация может поддерживать механизм для работы с ключом, но она не обязана реализовывать работу со всеми возможными параметрами. Для определения, какие параметры поддерживаются реализацией, могут быть использованы специальные объекты.

Наличие одного такого объекта в хранилище означает один набор поддерживаемых реализацией параметров. Для таких объектов значение атрибута CKA_CLASS установлено в CKO_DOMAIN_PARAMETERS.

Помимо общих атрибутов, объект должен содержать атрибут CKA_KEY_TYPE, значением которого должен быть тип ключа, для которого применимы эти параметры CKK_GOSTR3410 или CKK_GOSTR3410_512.

В обязательном атрибуте CKA_OBJECT_ID должен размещаться OID, определяющий параметры эллиптической кривой, представленный в кодировке ASN.1 в соответствии с [**ГОСТ Р ИСО/МЭК 8824-1]**.

Атрибут CKA_VALUE может отсутствовать или быть недоступным.

Пример использования объектов типа CKO_DOMAIN_PARAMETERS для определения возможности использования параметров эллиптической кривой приведен в приложении 3.1.

5.4 Механизмы хэширования.

Список механизмов хэширования расширяется для использования российского стандарта [**ГОСТ 34.11-2018].** Для двух функций хэширования, вводятся два новых механизма.

Стандарт [**ГОСТ 34.11-2018]** определяет две функции хэширования с длинами хэш-кода 256 и 512 бит. Для использования этих функций вводятся два механизма

```
CKM_GOSTR3411_2012_256
CKM_GOSTR3411_2012_512
```

с длинами хэш-кода 256 и 512 бит соответственно. Эти механизмы не используют параметры.

Механизмы используются в функции C_DigestInit в соответствии с **[PKCS#11_Base раздел 5.12.1].**

Примеры использования механизмов CKM_GOSTR3411_2012_512 и CKM_GOSTR3411_2012_256 находятся в приложениях 3.2 и 3.3 соответственно.

5.5 Механизмы HMAC.

Для вычисления MAC с помощью функции HMAC **[RFC2104]** на основе функции хэширования **[ГОСТ 34.11-2018]** вводятся механизмы

```
CKM_GOSTR3411_2012_256_HMAC
CKM_GOSTR3411_2012_512_HMAC
```

Механизм CKM_GOSTR3411_2012_256_HMAC соответствует функции HMAC_GOSTR3411_2012_256 **[ТК26АЛГ, раздел 4.1.1]**.

Механизм CKM_GOSTR3411_2012_512_HMAC соответсвтует функции HMAC_GOSTR3411_2012_512 **[ТК26АЛГ, раздел 4.1.2]**.

Эти механизмы не используют параметры.

Механизмы используются в функции C_SignInit. Совместно с этими механизмами допускается использование ключей типа CKK_GOST28147, CKK_MAGMA, CKK_KUZNECHIK или CKK_GENERIC_SECRET.

Примеры использования механизмов CKM_GOSTR3411_2012_512_HMAC и CKM_GOSTR3411_2012_256_HMAC находятся в приложениях 3.4 и 3.5 соответственно.

5.6 Механизмы PRF.

При реализации протокола TLS с росийскими алгоритмами в соответствии с документом [**ТК26TLS]** используются алгоритмы PRF **[RFC7836, раздел 4.2.1]**, основанные на стандарте хэширования [**ГОСТ 34.11-2018]**. Для получения значений псевдослучаных функций PRF вводятся механизмы

```
CKM_TLS_GOST_PRF_2012_256
CKM_TLS_GOST_PRF_2012_512
```

Механизм CKM_TLS_GOST_PRF_2012_256 соответствует функции PRF_TLS_GOSTR3411_2012_256 **[ТК26АЛГ, раздел 4.2.1.1]**.

Механизм CKM_TLS_GOST_PRF_2012_512 соответствует функции PRF_TLS_GOSTR3411_2012_512 **[ТК26АЛГ, раздел 4.2.1.2]**.

Механизмы используются в функции C_DeriveKey. Совместно с этими механизмами допускается использование ключей типа CKK_GOST28147, CKK_MAGMA, CKK_KUZNECHIK или CKK_GENERIC_SECRET.

Для задания параметров механизмов используется структура CK_TLS_PRF_PARAMS. Функция C_DeriveKey вырабатывает псевдослучайный набор байтов указанной в поле pulOutputLen длины, и возвращает результат не через

дескриптор ключа, а через поле pOutput. Аргумент phKey не используется и должен быть равен NULL_PTR.

Примеры использования механизмов приведены в приложениях 3.6 и 3.7.

5.7 Использование алгоритма хэширования ГОСТ 34.11-2018 в механизме генерации ключей PBKDF2.

Генерация ключей по алгоритму PKCS #5 PBKDF2 осуществляется в соответствии со стандартом **[PKCS#11_Mech, раздел 2.37]**. Функция C_GenerateKey вызывается с механизмом CKM_PKCS5_PBKD2. В качестве параметра механизма используется структура CK_PKCS5_PBKD2_PARAMS2.

При этом для использования алгоритма хэширования [**ГОСТ 34.11-2018]** в поле CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf структурыCK_PKCS5_PBKD2_PARAMS2 должно указываться значениеCKP_PKCS5_PBKD2_HMAC_GOSTR3411_2012_512.

Дополнительные параметры не используются, и значение поля pPrfData должно быть NULL, и значение поля ulPrfDataLen тоже должно быть 0.

Пример использования алгоритма хэширования ГОСТ 34.11-2018 в механизме генерации ключей PBKDF2приведен в приложении 3.8.

5.8 Механизм создания ключевой пары.

Для создания ключевой пары для стандарта [**ГОСТ 34.10-2018]** с типом ключа CKK_GOSTR3410_512 вводится механизм

CKM_GOSTR3410_512_KEY_PAIR_GEN

Механизм используется в функции C_GenerateKeyPair.

Механизм не использует параметров. Параметры эллиптической кривой задаются в шаблонах ключей в виде OID, представленного в кодировке ASN.1 в соответствии с [**ГОСТ Р ИСО/МЭК 8824-1]**, как описано в разделе «Наборы параметров». Если параметры не указаны, используется набор по умолчанию.

Пример генерации ключевой пары с длиной ключа ЭП 512 бит и соответствующего ключа проверки ЭП длиной 1024 бит алгоритма [**ГОСТ 34.10-2018]** приведены в приложении 3.9.

5.9 Механизм создания ключа проверки ЭП из ключа ЭП.

Для создания ключа проверки ЭП из ключа ЭП для стандарта [**ГОСТ 34.10-2018]** вводится механизм

`CKM_GOSTR3410_PUBLIC_KEY_DERIVE`

Механизм используется в функции `C_DeriveKey..`

Используются параметры эллиптической кривой, определенные в ключе ЭП.

Пример создания ключа проверки ЭП из ключа ЭПприведен в приложении 3.10.

5.10 Механизмы для подписи/проверки подписи.

Для реализации алгоритмов подписи/проверки подписи предвычисленного значения хэш-функции по алгоритму [**ГОСТ 34.10-2018]** на ключе ЭП длиной 512 бит вводится механизм

Механизм CKM_GOSTR3410_512 используется для реализации алгоритма подписи / проверки уже вычисленного значения хэш-функции на ключе длиной 512 бит

`CKM_GOSTR3410_512`

Для реализации совместого алгоритма хэширования **[ГОСТ 34.11-2018]** с длиной хэш-функции 256 бит и алгоритма подписи **[ГОСТ 34.10-2018]** на ключе ЭП длиной 256 бит вводится механизм

`CKM_GOSTR3410_WITH_GOSTR3411_2012_256`

Для реализации совместого алгоритма хэширования **[ГОСТ 34.11-2018]** с длиной хэш-функции 512 бит и алгоритма подписи **[ГОСТ 34.10-2018]** на ключе ЭП длиной 512 бит вводится механизм

`CKM_GOSTR3410_WITH_GOSTR3411_2012_512`

Механизмы используются в функциях `C_SignInit` и `C_VerifyInit,` параметры эллиптической кривой используются из ключа ЭП и явно в алгоритме не задаются.

Пример использования механизма подписи и проверки подписи приведены в приложениях 3.11 и 3.12 соответственно.

5.11 Механизм согласования ключей CKM_GOSTR3410_2012_DERIVE.

Для реализации протокола Диффи-Хеллмана для ключей **[ГОСТ 34.10-2018]** вводится механизм выработки производного ключа

CKM_GOSTR3410_2012_DERIVE

Механизм реализует алгоритм согласования ключей, на основе **[ГОСТ 34.10-2018]** и **[ГОСТ 34.11-2018]** и используется для согласования ключей **[ГОСТ 34.10-2018]** (256 и 512 бит). Он предназначен для получения ключевого материала длины 256 бит, использующегося в криптографических протоколах.

Механизм соответствует алгоритму VKO_GOSTR3410_2012_256 **[ТК26АЛГ, раздел 4.3.1]**.

К полученному таким образом ключу может быть применен (опционально) алгоритм диверсификации.

Параметры механизма CKM_GOSTR3410_2012_DERIVE задаются байтовым массивом, который имеет следующую структуру:

- 4 байта в little-endian формате представляют собой значение KDF. Значение определяет механизм диверсификации, один из следующих: CKD_NULL - нет диверсификации CKM_KDF_4357, CKD_CPDIVERSIFY_KDF, CKM_KDF_GOSTR3411_2012_256.
- 4 байта в little-endian формате задают длину ключа проверки ЭП в байтах (64 либо 128).
- 64 или 128 байт ключа проверки ЭП в little-endian формате.
- 4 байта в little-endian формате задают длину UKM (от 8 байт).
- UKM в little-endian формате.

Механизм используется в C_DeriveKey.

Исходными ключами для механизма могут служить ключи типа CKK_GOST3410, CKK_GOSTR3410_512.

В результате работы механизма допускается получение ключей типа CKK_GOST28147, CKK_MAGMA, CKK_KUZNECHIK или CKK_GENERIC_SECRET.

К механизму применяются следующие правила об извлекаемости и чувствительности результирующего ключа:

- Атрибуты CKA_SENSITIVE и CKA_EXTRACTABLE в шаблоне нового ключа могут принимать значения CK_TRUE или CK_FALSE. Если атрибуты не заданы, то они имеют значения по умолчанию, которые могут зависеть от устройства или других атрибутов (**[PKCS#11_Base раздел 4.10]**).
- Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет совпадать со значением атрибута CKA_SENSITIVE результирующего ключа.
- Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет противоположным значению атрибута CKA_EXTRACTABLE результирующего ключа.

Пример использования механизма CKM_GOSTR3410_2012_DERIVE находится в приложениях 3.13 и 3.14.

5.12 Механизм согласования ключей CKM_VKO_GOSTR3410_2012_512.

Для реализации протокола Диффи-Хеллмана для ключей **[ГОСТ 34.10-2018]** вводится механизм выработки производного ключа

CKM_VKO_GOSTR3410_2012_512

Механизм реализует алгоритм согласования ключей, на основе **[ГОСТ 34.10-2018]** и **[ГОСТ 34.11-2018]** и используется для согласования ключей **[ГОСТ 34.10-2018]** (512 бит) и предназначен для получения ключевого материала длины 512 бит, использующегося в криптографических протоколах.

Механизм соответствует алгоритму VKO_GOSTR3410_2012_512 **[ТК26АЛГ, раздел 4.3.2].**

Механизм использует один параметр – структуру типа CK_ECDH1_DERIVE_PARAMS. Поля этой структурызадают следующие параметры механизма:

- kdf – механизм деверсификации. В структуре не используется и должен быть установлен в CKD_NULL.
- ulSharedDataLen - длина используемой величины UKM в байтах (от 8 байт).
- pSharedData - указатель на величину UKM.
- ulPublicDataLen - длина ключа проверки ЭП в байтах (128 байт).
- pPublicData - ключ проверки ЭП.

Механизм используется в функции C_DeriveKey.

Исходными ключами для механизма могут служить ключи типа CKK_GOSTR3410_512.

В результате работы механизма допускается получение ключей типа CKK_MAGMA_TWIN_KEY, CKK_KUZNECHIK_TWIN_KEY или CKK_GENERIC_SECRET.

К механизму применяются следующие правила об извлекаемости и чувствительности результирующего ключа:

- Атрибуты CKA_SENSITIVE и CKA_EXTRACTABLE в шаблоне нового ключа могут принимать значения CK_TRUE или CK_FALSE. Если атрибуты не заданы, то они имеют значения по умолчанию, которые могут зависеть от устройства или других атрибутов (**[PKCS#11_Base раздел 4.10]**).
- Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет совпадать со значением атрибута CKA_SENSITIVE результирующего ключа.Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет противоположным значению атрибута CKA_EXTRACTABLE результирующего ключа.

Пример использования механизма CKM_VKO_GOSTR3410_2012_512 приведен в приложении 3.15.

5.13 Механизм согласования ключей CKM_GOST_KEG.

Для реализации протокола Диффи-Хеллмана для ключей **[ГОСТ 34.10-2018]** вводится механизм выработки производного ключа

CKM_GOST_KEG

Механизм реализует алгоритм согласования ключей, на основе **[ГОСТ 34.10-2018]** и **[ГОСТ 34.11-2018]** и используется для согласования ключей **[ГОСТ 34.10-2018]** (256 бит и 512 бит) и предназначен для получения ключевого материала длины 512 бит, использующегося в криптографических протоколах.

Механизм соответствует алгоритму KEG **[ТК26TLS_1.2, раздел 6.4.5.1].**

Механизм использует один параметр – структуру типа CK_ECDH1_DERIVE_PARAMS. Поля этой структурызадают следующие параметры механизма:

- kdf – механизм деверсификации. В структуре не используется и должен быть установлен в CKD_NULL.
- ulSharedDataLen – длина используемой величины UKM в байтах.
- pSharedData - указатель на величину UKM.
- ulPublicDataLen - длина ключа проверки ЭП в байтах (64 или 128 байт).
- pPublicData - ключ проверки ЭП.

Механизм используется в функции C_DeriveKey.

Исходными ключами для механизма могут служить ключи типа CKK_GOSTR3410 и CKK_GOSTR3410_512.

В результате работы механизма допускается получение ключей типа CKK_MAGMA_TWIN_KEY, CKK_KUZNECHIK_TWIN_KEY или CKK_GENERIC_SECRET.

К механизму применяются следующие правила об извлекаемости и чувствительности результирующего ключа:

- Атрибуты CKA_SENSITIVE и CKA_EXTRACTABLE в шаблоне нового ключа могут принимать значения CK_TRUE или CK_FALSE. Если атрибуты не заданы, то они имеют значения по умолчанию, которые могут зависеть от устройства или других атрибутов (**[PKCS#11_Base раздел 4.10]**).

- Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет совпадать со значением атрибута CKA_SENSITIVE результирующего ключа.
- Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет противоположным значению атрибута CKA_EXTRACTABLE результирующего ключа.

Пример использования механизма CKM_GOST_KEG приведен в приложениях 3.16 и 3.17.

5.14 Механизм согласования ключей CKM_ECDH1_DERIVE.

Для одной из модификаций протокола Диффи-Хеллмана для ключей **[ГОСТ 34.10-2018]** вводится механизм выработки производного ключа

CKM_ECDH1_DERIVE

Механизм реализует алгоритм согласования ключей, на основе **[ГОСТ 34.10-2018]** и используется для согласования ключей **[ГОСТ 34.10-2018]** (256 бит и 512 бит) и предназначен для получения ключевого материала длины 256 бит, использующегося в криптографических протоколах.

Механизм соответствует алгоритму выработки общего секретного значения ECDHE **[ТК26TLS_1.3, раздел 8.5].**

Механизм использует один параметр – структуру типа CK_ECDH1_DERIVE_PARAMS. Поля этой структуры задают следующие параметры механизма:

- kdf – механизм деверсификации. В структуре не используется и должен быть установлен в CKD_NULL.

- ulSharedDataLen – длина используемой величины UKM в байтах. В структуре не используется и должен быть установлен в 0.
- pSharedData - указатель на величину UKM. В структуре не используется и должен быть установлен в NULL.
- ulPublicDataLen - длина ключа проверки ЭП в байтах (64 или 128 байт).
- pPublicData - ключ проверки ЭП.

Механизм используется в функции C_DeriveKey.

Исходными ключами для механизма могут служить ключи типа CKK_GOSTR3410_256 и CKK_GOSTR3410_512.

Если в качестве исходных ключей используются ключи типа CKK_GOSTR3410_256, то в результате могут быть получены ключи типа CKK_MAGMA, CKK_KUZNECHIK или CKK_GENERIC_SECRET.

Если в качестве исходных ключей используются ключи типа CKK_GOSTR3410_512, то в результате могут быть получены ключи типа CKK_MAGMA_TWIN_KEY, CKK_KUZNECHIK_TWIN_KEY или CKK_GENERIC_SECRET.

К механизму применяются стандартные правила об извлекаемости и чувствительности результирующего ключа, определенные в **[PKCS#11_Mech, пункт 2.3.18]** следующим образом:

- Атрибуты CKA_SENSITIVE и CKA_EXTRACTABLE в шаблоне нового ключа могут принимать значения CK_TRUE или CK_FALSE. Если атрибуты не заданы, то они имеют значения по умолчанию, которые могут зависеть от устройства или других атрибутов (**[PKCS#11_Base раздел 4.10]**).
- Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_ALWAYS_SENSITIVE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута результирующего ключа будет совпадать со значением атрибута CKA_SENSITIVE результирующего ключа.
- Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_FALSE, то соответствующий атрибут результирующего ключа также будет иметь значение CK_FALSE. Если атрибут CKA_NEVER_EXTRACTABLE исходного ключа имеет значение CK_TRUE, то значение соответствующего атрибута

результирующего ключа будет противоположным значению атрибута CKA_EXTRACTABLE результирующего ключа.

Пример использования механизма CKM_ECDH1_DERIVE приведен в приложениях 3.18 и 3.19.

1. Приложение 1 (обязательное). Заголовочный файл со списком используемых идентификаторов.

```
#ifndef CK_VENDOR_PKCS11_RU_TEAM_TC26
#define NSSCK_VENDOR_PKCS11_RU_TEAM           0xD4321000UL /*0x80000000 | 0x54321000*/
#define CK_VENDOR_PKCS11_RU_TEAM_TC26         NSSCK_VENDOR_PKCS11_RU_TEAM
#endif // CK_VENDOR_PKCS11_RU_TEAM_TC26

typedef struct CK_KDF_TREE_GOST_PARAMS {
    CK_ULONG ulLabelLength;
    CK_BYTE_PTR pLabel;
    CK_ULONG ulSeedLength;
    CK_BYTE_PTR pSeed;
    CK_ULONG ulR;
    CK_ULONG ulL;
    CK_ULONG ulOffset;
} CK_KDF_TREE_GOST_PARAMS;


/* GOST KEY TYPES */

#define CKK_KUZNECHIK                         (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x004UL)
#define CKK_KUZNYECHIK                        CKK_KUZNECHIK
#define CKK_MAGMA                             (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x005UL)
#define CKK_KUZNECHIK_TWIN_KEY                (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x006UL)
#define CKK_KUZNYECHIK_TWIN_KEY               CKK_KUZNECHIK_TWIN_KEY
#define CKK_MAGMA_TWIN_KEY                    (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x007UL)

#define CKK_GOSTR3410_256                     CKK_GOSTR3410
#define CKK_GOSTR3410_512                     (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x003UL)

/* GOST OBJECT ATTRIBUTES */
#define CKA_GOSTR3410_256PARAMS               CKA_GOSTR3410_PARAMS

/* PKCS #5 PRF Functions */
#define     CKP_PKCS5_PBKD2_HMAC_GOSTR3411_2012_512     (CK_VENDOR_PKCS11_RU_TEAM_TC26
|0x003UL)

/* GOST MECHANISMS */
#define CKM_KDF_HMAC3411_2012_256             (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x028UL)
#define KDF_HMAC3411_2012_256                 CKM_KDF_HMAC3411_2012_256
// deprecated
//#define CKM_KDF_GOSTR3411_EXPORT            (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x029UL)
#define CKM_KDF_TREE_GOSTR3411_2012_256       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x02AUL)
#define KDF_TREE_GOSTR3411_2012_256           CKM_KDF_TREE_GOSTR3411_2012_256

#define CKM_KUZNECHIK_KEXP_15_WRAP            (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x02BUL)
#define CKM_KUZNYECHIK_KEXP_15_WRAP           CKM_KUZNECHIK_KEXP_15_WRAP
#define CKM_MAGMA_KEXP_15_WRAP                (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x02CUL)
#define CKM_KUZNECHIK_MGM                     (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x02DUL)
#define CKM_KUZNYECHIK_MGM                    CKM_KUZNECHIK_MGM
#define CKM_MAGMA_MGM                         (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x02EUL)

#define CKM_KUZNECHIK_KEY_GEN                 (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x030UL)
#define CKM_KUZNYECHIK_KEY_GEN                CKM_KUZNECHIK_KEY_GEN
#define CKM_KUZNECHIK_ECB                     (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x031UL)
#define CKM_KUZNYECHIK_ECB                    CKM_KUZNECHIK_ECB
#define CKM_KUZNECHIK_CTR_ACPKM               (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x032UL)
#define CKM_KUZNYECHIK_CTR_ACPKM              CKM_KUZNECHIK_CTR_ACPKM
#define CKM_KUZNECHIK_MAC                     (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x033UL)
#define CKM_KUZNYECHIK_MAC                    CKM_KUZNECHIK_MAC

#define CKM_MAGMA_KEY_GEN                     (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x034UL)
#define CKM_MAGMA_ECB                         (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x035UL)
#define CKM_MAGMA_CTR_ACPKM                   (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x036UL)
#define CKM_MAGMA_MAC                         (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x037UL)


#define CKM_GOSTR3410_256_KEY_PAIR_GEN        CKM_GOSTR3410_KEY_PAIR_GEN
```

```
#define CKM_GOSTR3410_512_KEY_PAIR_GEN        (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x005UL)
#define CKM_GOSTR3410_256                     CKM_GOSTR3410
#define CKM_GOSTR3410_512                     (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x006UL)
#define CKM_GOSTR3410_2012_DERIVE             (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x007UL)
#define CKM_GOSTR3410_12_DERIVE               CKM_GOSTR3410_2012_DERIVE
#define CKM_GOSTR3410_WITH_GOSTR3411_94       CKM_GOSTR3410_WITH_GOSTR3411
#define CKM_GOSTR3410_WITH_GOSTR3411_2012_256 (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x008UL)
#define CKM_GOSTR3410_WITH_GOSTR3411_2012_512 (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x009UL)
#define CKM_GOSTR3410_WITH_GOSTR3411_12_256   CKM_GOSTR3410_WITH_GOSTR3411_2012_256
#define CKM_GOSTR3410_WITH_GOSTR3411_12_512   CKM_GOSTR3410_WITH_GOSTR3411_2012_512
#define CKM_GOSTR3410_PUBLIC_KEY_DERIVE       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x00AUL)
#define CKM_GOSTR3410_512_PUBLIC_KEY_DERIVE   (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x00BUL)

#define CKM_GOSTR3411_94                      CKM_GOSTR3411
#define CKM_GOSTR3411_2012_256                (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x012UL)
#define CKM_GOSTR3411_2012_512                (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x013UL)
#define CKM_GOSTR3411_12_256                  CKM_GOSTR3411_2012_256
#define CKM_GOSTR3411_12_512                  CKM_GOSTR3411_2012_512
#define CKM_GOSTR3411_94_HMAC                 CKM_GOSTR3411_HMAC
#define CKM_GOSTR3411_2012_256_HMAC           (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x014UL)
#define CKM_GOSTR3411_2012_512_HMAC           (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x015UL)
#define CKM_GOSTR3411_12_256_HMAC             CKM_GOSTR3411_2012_256_HMAC
#define CKM_GOSTR3411_12_512_HMAC             CKM_GOSTR3411_2012_512_HMAC

#define CKM_TLS_GOST_PRF_2012_256             (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x016UL)
#define CKM_TLS_GOST_PRF_2012_512             (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x017UL)
#define CKM_TLS_GOST_PRE_MASTER_KEY_GEN       CKM_GOST28147_KEY_GEN
#define CKM_TLS_GOST_MASTER_KEY_DERIVE_2012_256 (CK_VENDOR_PKCS11_RU_TEAM_TC26|0x018UL)

#define CKM_KDF_4357                          (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x025UL)
#define CKM_KDF_GOSTR3411_2012_256            (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x026UL)
#define CKM_VKO_GOSTR3410_2012_512            (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x038UL)
#define CKM_GOST_KEG                          (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x039UL)
```

2. **Приложение 2 (справочное). Контрольные примеры использования ГОСТ 34.12-2018, ГОСТ 34.13-2018.**

Приложение 2.1 (справочное). Пример использования механизма CKM_KUZNECHIK_KEY_GEN.

```
CK_RV   sample_generate_key_kuznechik(CK_FUNCTION_LIST_PTR   pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;

    CK_MECHANISM mechanism = { CKM_KUZNECHIK_KEY_GEN, NULL, 0 };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_KUZNECHIK;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;

    CK_ATTRIBUTE keyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_EXTRACTABLE, &bTrue,        sizeof(bTrue)},
         {CKA_SENSITIVE,   &bFalse,       sizeof(bFalse)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_DECRYPT,     &bTrue,        sizeof(bTrue)},
    };

    rv  =  pF->C_GenerateKey(hSession,  &mechanism,  keyTemplate,
sizeof(keyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);

    CK_ATTRIBUTE keyAttribute = { CKA_VALUE, NULL, 0 };
    rv     =     pF->C_GetAttributeValue(hSession,     keyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);
    assert(keyAttribute.ulValueLen == 32);

    rv = pF->C_DestroyObject(hSession, keyHandle);
    assert(rv == CKR_OK);

    return CKR_OK;
}
```

Приложение 2.2 (справочное). Пример использования механизма CKM_KUZNECHIK_ECB.

```
CK_RV    sample_encrypt_kuznechik_ecb(CK_FUNCTION_LIST_PTR    pF,
CK_SESSION_HANDLE hSession)
{
```

```
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;

    CK_MECHANISM encryptMechanism = { CKM_KUZNECHIK_ECB, NULL, 0
};
    CK_BYTE sourceText[] =
    {
        0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00,
        0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A,
        0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
        0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00,
        0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
        0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00, 0x11,
    };
    CK_BYTE ETALON[] = {
        0x7F, 0x67, 0x9D, 0x90, 0xBE, 0xBC, 0x24, 0x30,
        0x5A, 0x46, 0x8D, 0x42, 0xB9, 0xD4, 0xED, 0xCD,
        0xB4, 0x29, 0x91, 0x2C, 0x6E, 0x00, 0x32, 0xF9,
        0x28, 0x54, 0x52, 0xD7, 0x67, 0x18, 0xD0, 0x8B,
        0xF0, 0xCA, 0x33, 0x54, 0x9D, 0x24, 0x7C, 0xEE,
        0xF3, 0xF5, 0xA5, 0x31, 0x3B, 0xD4, 0xB1, 0x57,
        0xD0, 0xB0, 0x9C, 0xCD, 0xE8, 0x30, 0xB9, 0xEB,
        0x3A, 0x02, 0xC4, 0xC5, 0xAA, 0x8A, 0xDA, 0x98,
    };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_KUZNECHIK;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
    CK_BYTE sourceKeyValue[] =
    {
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10,
        0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF,
    };

    CK_ATTRIBUTE keyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bFalse,       sizeof(bFalse)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,      sourceKeyValue,sizeof(sourceKeyValue)},
    };


    rv      =      pF->C_CreateObject(hSession,      keyTemplate,
sizeof(keyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);
```

```
    rv    =    pF->C_EncryptInit(hSession,    &encryptMechanism,
keyHandle);
    assert(rv == CKR_OK);

    CK_BYTE encryptText[sizeof(sourceText)] = { 0 };
    CK_ULONG encryptTextSize = sizeof(encryptText);
    rv = pF->C_Encrypt(hSession, sourceText, sizeof(sourceText),
encryptText, &encryptTextSize);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, keyHandle);
    assert(rv == CKR_OK);
    return   memcmp(encryptText,   ETALON,   sizeof(ETALON))   ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.3 (справочное). Пример использования механизма CKM_KUZNECHIK_CTR_ACPKM.

```
CK_RV    sample_encrypt_kuznechik_ctr(CK_FUNCTION_LIST_PTR    pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;

    // Преполагаем, что LE система
    CK_BYTE mechanismParam[] =
    {
        // meshing period
        0x00, 0x00, 0x20, 0x00,
        // iv
        0x12, 0x34, 0x56, 0x78, 0x90, 0xAB, 0xCE, 0xF0,
    };

    CK_MECHANISM  encryptMechanism  =  {  CKM_KUZNECHIK_CTR_ACPKM,
mechanismParam, sizeof(mechanismParam) };
    CK_BYTE sourceText[] =
    {
        0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00,
        0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A,
        0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
        0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00,
        0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
        0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00, 0x11,
    };
    CK_BYTE ETALON[] = {
        0xF1, 0x95, 0xD8, 0xBE, 0xC1, 0x0E, 0xD1, 0xDB,
        0xD5, 0x7B, 0x5F, 0xA2, 0x40, 0xBD, 0xA1, 0xB8,
        0x85, 0xEE, 0xE7, 0x33, 0xF6, 0xA1, 0x3E, 0x5D,
        0xF3, 0x3C, 0xE4, 0xB3, 0x3C, 0x45, 0xDE, 0xE4,
        0xA5, 0xEA, 0xE8, 0x8B, 0xE6, 0x35, 0x6E, 0xD3,
```

```
        0xD5, 0xE8, 0x77, 0xF1, 0x35, 0x64, 0xA3, 0xA5,
        0xCB, 0x91, 0xFA, 0xB1, 0xF2, 0x0C, 0xBA, 0xB6,
        0xD1, 0xC6, 0xD1, 0x58, 0x20, 0xBD, 0xBA, 0x73,
    };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_KUZNECHIK;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
    CK_BYTE sourceKeyValue[] =
    {
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10,
        0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF,
    };

    CK_ATTRIBUTE keyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bFalse,       sizeof(bFalse)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,      sourceKeyValue,sizeof(sourceKeyValue)},
    };


    rv      =      pF->C_CreateObject(hSession,      keyTemplate,
sizeof(keyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);

    rv    =    pF->C_EncryptInit(hSession,    &encryptMechanism,
keyHandle);
    assert(rv == CKR_OK);

    CK_BYTE encryptText[sizeof(sourceText)] = { 0 };
    CK_ULONG encryptTextSize = sizeof(encryptText);
    rv = pF->C_Encrypt(hSession, sourceText, sizeof(sourceText),
encryptText, &encryptTextSize);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, keyHandle);
    assert(rv == CKR_OK);
    return   memcmp(encryptText,   ETALON,   sizeof(ETALON))   ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.4 (справочное). Пример использования механизма CKM_KUZNECHIK_MAC.

```
CK_RV        sample_mac_kuznechik(CK_FUNCTION_LIST_PTR        pF,
CK_SESSION_HANDLE hSession)
{
```

```
    CK_RV rv = CKR_OK;
    CK_MECHANISM mechanism = { CKM_KUZNECHIK_MAC, NULL_PTR, 0 };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_KUZNECHIK;
    CK_OBJECT_CLASS keyObject = CKO_SECRET_KEY;
    CK_BYTE keyValue[] = {
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10,
        0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF,
    };
    CK_ATTRIBUTE secretKeyTemplate[] = {
         {CKA_CLASS,          &keyObject,  sizeof(keyObject)},
         {CKA_KEY_TYPE,       &keyType,    sizeof(keyType)},
         {CKA_TOKEN,          &bFalse,     sizeof(bTrue)},
         {CKA_SIGN,           &bTrue,      sizeof(bTrue)},
         {CKA_VALUE,          keyValue,    sizeof(keyValue)},
    };

    CK_BYTE testData[] = {
        0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00,
        0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A,
        0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
        0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00,
        0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
        0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00, 0x11,
    };
    const CK_BYTE ETALON[] = {
        0x33, 0x6F, 0x4D, 0x29, 0x60, 0x59, 0xFB, 0xE3,
        0x4D, 0xDE, 0xB3, 0x5B, 0x37, 0x74, 0x9C, 0x67,
    };
    CK_BYTE value[sizeof(ETALON)];
    CK_ULONG valueLength = sizeof(value);
    CK_OBJECT_HANDLE secretKey = CK_INVALID_HANDLE;

    rv    =    pF->C_CreateObject(hSession,    secretKeyTemplate,
sizeof(secretKeyTemplate) / sizeof(CK_ATTRIBUTE), &secretKey);
    assert(rv == CKR_OK);

    rv = pF->C_SignInit(hSession, &mechanism, secretKey);
    assert(rv == CKR_OK);
    rv = pF->C_Sign(hSession, testData, sizeof(testData), value,
&valueLength);
    assert(rv == CKR_OK);
    assert(valueLength == sizeof(value));

    rv = pF->C_DestroyObject(hSession, secretKey);
    assert(rv == CKR_OK);
```

```
    return      memcmp(value,      ETALON,      valueLength)      ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.5 (справочное). Пример использования механизма CKM_KUZNECHIK_KEXP_15_WRAP.

```
CK_RV       sample_kexp15_kuznechik(CK_FUNCTION_LIST_PTR       pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE kekKeyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE cekKeyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE unwrappedKeyHandle = CK_INVALID_HANDLE;

    CK_BYTE iv[] = {
        0x09, 0x09, 0x47, 0x2D, 0xD9, 0xF2, 0x6B, 0xE8,
    };
    CK_MECHANISM kexpMechanism = { CKM_KUZNECHIK_KEXP_15_WRAP, iv,
sizeof(iv) };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_KUZNECHIK;
    CK_KEY_TYPE twinKeyType = CKK_KUZNECHIK_TWIN_KEY;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;

    CK_BYTE cekKeyValue[] = {
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10,
        0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF,
    };
    CK_ATTRIBUTE cekKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_DECRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,       cekKeyValue,   sizeof(cekKeyValue)},
    };

    CK_BYTE kekKeyValue[] = {
         0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
         0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
         0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
         0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
         0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
         0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
         0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    };
```

```
    CK_ATTRIBUTE kekKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &twinKeyType,  sizeof(twinKeyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_WRAP,        &bTrue,        sizeof(bTrue)},
         {CKA_UNWRAP,      &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,       kekKeyValue,   sizeof(kekKeyValue)},
    };

    CK_ATTRIBUTE unwrappedKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_EXTRACTABLE, &bTrue,        sizeof(bTrue)},
         {CKA_SENSITIVE,   &bFalse,       sizeof(bFalse)},
    };

    CK_BYTE ETALON[] = {
         0xE3, 0x61, 0x84, 0xE8, 0x4E, 0x8D, 0x73, 0x6F,
         0xF3, 0x6C, 0xC2, 0xE5, 0xAE, 0x06, 0x5D, 0xC6,
         0x56, 0xB2, 0x3C, 0x20, 0xF5, 0x49, 0xB0, 0x2F,
         0xDF, 0xF8, 0x8E, 0x1F, 0x3F, 0x30, 0xD8, 0xC2,
         0x9A, 0x53, 0xF3, 0xCA, 0x55, 0x4D, 0xBA, 0xD8,
         0x0D, 0xE1, 0x52, 0xB9, 0xA4, 0x62, 0x5B, 0x32,
    };

    rv     =     pF->C_CreateObject(hSession,     cekKeyTemplate,
sizeof(cekKeyTemplate) / sizeof(CK_ATTRIBUTE), &cekKeyHandle);
    assert(rv == CKR_OK);
    rv     =     pF->C_CreateObject(hSession,     kekKeyTemplate,
sizeof(kekKeyTemplate) / sizeof(CK_ATTRIBUTE), &kekKeyHandle);
    assert(rv == CKR_OK);

    CK_BYTE value[sizeof(ETALON)];
    CK_ULONG valueLength = sizeof(value);
    rv  =  pF->C_WrapKey(hSession,  &kexpMechanism,  kekKeyHandle,
cekKeyHandle, NULL, &valueLength);
    assert(rv == CKR_OK);
    assert(valueLength == sizeof(ETALON));
    rv  =  pF->C_WrapKey(hSession,  &kexpMechanism,  kekKeyHandle,
cekKeyHandle, value, &valueLength);
    assert(rv == CKR_OK);
    assert(valueLength == sizeof(ETALON));
    assert(memcmp(value, ETALON, valueLength) == 0);

    rv = pF->C_UnwrapKey(hSession, &kexpMechanism, kekKeyHandle,
ETALON, sizeof(ETALON),
        unwrappedKeyTemplate,   sizeof(unwrappedKeyTemplate)   /
sizeof(CK_ATTRIBUTE), &unwrappedKeyHandle);
    assert(rv == CKR_OK);
```

```
    CK_BYTE unwrappedKeyValue[sizeof(cekKeyValue)];
    CK_ATTRIBUTE keyAttribute = { CKA_VALUE, unwrappedKeyValue,
sizeof(unwrappedKeyValue) };
    rv = pF->C_GetAttributeValue(hSession, unwrappedKeyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);
    assert(keyAttribute.ulValueLen == sizeof(cekKeyValue));

    rv = pF->C_DestroyObject(hSession, unwrappedKeyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, kekKeyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, cekKeyHandle);
    assert(rv == CKR_OK);

    return         memcmp(unwrappedKeyValue,         cekKeyValue,
sizeof(cekKeyValue)) ? CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.6 (справочное). Пример использования механизма CKM_KUZNECHIK_MGM.

```
CK_RV        sample_kuznechik_mgm(CK_FUNCTION_LIST_PTR        pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_BYTE pIv[] = {
            0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00,
            0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
    };
    CK_BYTE addData[] = {
            0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
            0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
            0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
            0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
            0xEA, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
            0x05,
    };
    CK_BYTE plainText[] = {
            0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00,
            0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
            0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
            0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A,
            0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
            0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00,
            0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
            0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00, 0x11,
            0xAA, 0xBB, 0xCC,
    };
    CK_BYTE keyValue[] = {
            0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF,
```

```
            0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
            0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10,
            0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF,
    };
    CK_BYTE etalonText[] = {
            0xA9, 0x75, 0x7B, 0x81, 0x47, 0x95, 0x6E, 0x90,
            0x55, 0xB8, 0xA3, 0x3D, 0xE8, 0x9F, 0x42, 0xFC,
            0x80, 0x75, 0xD2, 0x21, 0x2B, 0xF9, 0xFD, 0x5B,
            0xD3, 0xF7, 0x06, 0x9A, 0xAD, 0xC1, 0x6B, 0x39,
            0x49, 0x7A, 0xB1, 0x59, 0x15, 0xA6, 0xBA, 0x85,
            0x93, 0x6B, 0x5D, 0x0E, 0xA9, 0xF6, 0x85, 0x1C,
            0xC6, 0x0C, 0x14, 0xD4, 0xD3, 0xF8, 0x83, 0xD0,
            0xAB, 0x94, 0x42, 0x06, 0x95, 0xC7, 0x6D, 0xEB,
            0x2C, 0x75, 0x52,
    };
    CK_BYTE etalonTag[] = {
            0xCF, 0x5D, 0x65, 0x6F, 0x40, 0xC3, 0x4F, 0x5C,
            0x46, 0xE8, 0xBB, 0x0E, 0x29, 0xFC, 0xDB, 0x4C,
    };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_KUZNECHIK;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
    CK_ATTRIBUTE secretKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_DECRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,       keyValue,      sizeof(keyValue)},
    };
    CK_ULONG secretKeyTemplateSize = sizeof(secretKeyTemplate) /
sizeof(*secretKeyTemplate);

    byte actualValue[sizeof(plainText) + sizeof(etalonTag)];
    byte decrypted[sizeof(plainText)];

    CK_GCM_PARAMS mechanismParam = {
         pIv, sizeof(pIv), sizeof(pIv) * 8,
         addData, sizeof(addData),
         sizeof(etalonTag) * 8 };
    CK_MECHANISM mechanism = { CKM_KUZNECHIK_MGM, &mechanismParam,
sizeof(mechanismParam) };

    CK_OBJECT_HANDLE secretKey = CK_INVALID_HANDLE;
    rv    =    pF->C_CreateObject(hSession,    secretKeyTemplate,
secretKeyTemplateSize, &secretKey);
    assert(rv == CKR_OK);

    rv = pF->C_EncryptInit(hSession, &mechanism, secretKey);
    assert(rv == CKR_OK);
```

```
    CK_ULONG result = sizeof(actualValue);
    rv = pF->C_Encrypt(hSession, plainText, sizeof(plainText),
actualValue, &result);
    assert(rv == CKR_OK);
    assert(result == sizeof(actualValue));

    rv = pF->C_DecryptInit(hSession, &mechanism, secretKey);
    assert(rv == CKR_OK);
    result = sizeof(decrypted);
    rv = pF->C_Decrypt(hSession, actualValue,
sizeof(actualValue), decrypted, &result);
    assert(rv == CKR_OK);
    assert(result == sizeof(decrypted));

    rv = pF->C_DestroyObject(hSession, secretKey);
    assert(rv == CKR_OK);

    assert(memcmp(etalonText, actualValue, sizeof(etalonText)) ==
0);
    assert(memcmp(plainText, decrypted, sizeof(plainText)) == 0);
    assert(memcmp(etalonTag, actualValue + sizeof(plainText),
sizeof(etalonTag)) == 0);
    return CKR_OK;
}
```

Приложение 2.7 (справочное). Пример использования механизма CKM_MAGMA_KEY_GEN.

```
CK_RV     sample_generate_key_magma(CK_FUNCTION_LIST_PTR     pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;

    CK_MECHANISM mechanism = { CKM_MAGMA_KEY_GEN, NULL, 0 };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_MAGMA;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;

    CK_ATTRIBUTE keyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_EXTRACTABLE, &bTrue,        sizeof(bTrue)},
         {CKA_SENSITIVE,   &bFalse,       sizeof(bFalse)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_DECRYPT,     &bTrue,        sizeof(bTrue)},
    };
```

```
    rv = pF->C_GenerateKey(hSession, &mechanism, keyTemplate,
sizeof(keyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);

    CK_ATTRIBUTE keyAttribute = { CKA_VALUE, NULL, 0 };
    rv = pF->C_GetAttributeValue(hSession, keyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);
    assert(keyAttribute.ulValueLen == 32);

    rv = pF->C_DestroyObject(hSession, keyHandle);
    assert(rv == CKR_OK);

    return CKR_OK;
}
```

Приложение 2.8 (справочное). Пример использования механизма CKM_MAGMA_ECB.

```
CK_RV      sample_encrypt_magma_ecb(CK_FUNCTION_LIST_PTR      pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;

    CK_MECHANISM encryptMechanism = { CKM_MAGMA_ECB, NULL, 0 };
    CK_BYTE sourceText[] =
    {
        0x92, 0xde, 0xf0, 0x6b, 0x3c, 0x13, 0x0a, 0x59,
        0xdb, 0x54, 0xc7, 0x04, 0xf8, 0x18, 0x9d, 0x20,
        0x4a, 0x98, 0xfb, 0x2e, 0x67, 0xa8, 0x02, 0x4c,
        0x89, 0x12, 0x40, 0x9b, 0x17, 0xb5, 0x7e, 0x41,
    };
    CK_BYTE ETALON[] = {
        0x2B, 0x07, 0x3F, 0x04, 0x94, 0xF3, 0x72, 0xA0,
        0xDE, 0x70, 0xE7, 0x15, 0xD3, 0x55, 0x6E, 0x48,
        0x11, 0xD8, 0xD9, 0xE9, 0xEA, 0xCF, 0xBC, 0x1E,
        0x7C, 0x68, 0x26, 0x09, 0x96, 0xC6, 0x7E, 0xFB,
    };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_MAGMA;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
    CK_BYTE sourceKeyValue[] =
    {
        0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
        0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
        0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
        0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
    };

    CK_ATTRIBUTE keyTemplate[] = {
```

```
        {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
        {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
        {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
        {CKA_PRIVATE,     &bFalse,       sizeof(bFalse)},
        {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
        {CKA_VALUE,      sourceKeyValue,sizeof(sourceKeyValue)},
    };


    rv      =      pF->C_CreateObject(hSession,      keyTemplate,
sizeof(keyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);

    rv    =    pF->C_EncryptInit(hSession,    &encryptMechanism,
keyHandle);
    assert(rv == CKR_OK);

    CK_BYTE encryptText[sizeof(sourceText)] = { 0 };
    CK_ULONG encryptTextSize = sizeof(encryptText);
    rv = pF->C_Encrypt(hSession, sourceText, sizeof(sourceText),
encryptText, &encryptTextSize);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, keyHandle);
    assert(rv == CKR_OK);
    return   memcmp(encryptText,   ETALON,   sizeof(ETALON))   ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.9 (справочное). Пример использования механизма CKM_MAGMA_CTR_ACPKM.

```
CK_RV      sample_encrypt_magma_ctr(CK_FUNCTION_LIST_PTR      pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;

    // Преполагаем, что LE система
    CK_BYTE mechanismParam[] =
    {
        // meshing period
        0x00, 0x00, 0x04, 0x00,
        // iv
        0x12, 0x34, 0x56, 0x78,
    };

    CK_MECHANISM   encryptMechanism   =   {   CKM_MAGMA_CTR_ACPKM,
mechanismParam, sizeof(mechanismParam) };
    CK_BYTE sourceText[] =
    {
        0x92, 0xde, 0xf0, 0x6b, 0x3c, 0x13, 0x0a, 0x59,
        0xdb, 0x54, 0xc7, 0x04, 0xf8, 0x18, 0x9d, 0x20,
```

```
        0x4a, 0x98, 0xfb, 0x2e, 0x67, 0xa8, 0x02, 0x4c,
        0x89, 0x12, 0x40, 0x9b, 0x17, 0xb5, 0x7e, 0x41,
    };
    CK_BYTE ETALON[] = {
        0x4E, 0x98, 0x11, 0x0C, 0x97, 0xB7, 0xB9, 0x3C,
        0x3E, 0x25, 0x0D, 0x93, 0xD6, 0xE8, 0x5D, 0x69,
        0x13, 0x6D, 0x86, 0x88, 0x07, 0xB2, 0xDB, 0xEF,
        0x56, 0x8E, 0xB6, 0x80, 0xAB, 0x52, 0xA1, 0x2D,
    };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_MAGMA;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
    CK_BYTE sourceKeyValue[] =
    {
        0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
        0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
        0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
        0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
    };

    CK_ATTRIBUTE keyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bFalse,       sizeof(bFalse)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,      sourceKeyValue,sizeof(sourceKeyValue)},
    };


    rv      =      pF->C_CreateObject(hSession,      keyTemplate,
sizeof(keyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);

    rv    =    pF->C_EncryptInit(hSession,    &encryptMechanism,
keyHandle);
    assert(rv == CKR_OK);

    CK_BYTE encryptText[sizeof(sourceText)] = { 0 };
    CK_ULONG encryptTextSize = sizeof(encryptText);
    rv = pF->C_Encrypt(hSession, sourceText, sizeof(sourceText),
encryptText, &encryptTextSize);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, keyHandle);
    assert(rv == CKR_OK);
    return   memcmp(encryptText,   ETALON,   sizeof(ETALON))   ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.10 (справочное). Пример использования механизма CKM_MAGMA_MAC.

```
CK_RV sample_mac_magma(CK_FUNCTION_LIST_PTR pF, CK_SESSION_HANDLE
hSession)
{
    CK_RV rv = CKR_OK;
    CK_MECHANISM mechanism = { CKM_MAGMA_MAC, NULL_PTR, 0 };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_MAGMA;
    CK_OBJECT_CLASS keyObject = CKO_SECRET_KEY;
    CK_BYTE keyValue[] = {
        0xff, 0xee, 0xdd, 0xcc, 0xbb, 0xaa, 0x99, 0x88,
        0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
        0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
        0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff,
    };
    CK_ATTRIBUTE secretKeyTemplate[] = {
         {CKA_CLASS,          &keyObject,  sizeof(keyObject)},
         {CKA_KEY_TYPE,       &keyType,    sizeof(keyType)},
         {CKA_TOKEN,          &bFalse,     sizeof(bTrue)},
         {CKA_SIGN,           &bTrue,      sizeof(bTrue)},
         {CKA_VALUE,          keyValue,    sizeof(keyValue)},
    };

    CK_BYTE testData[] = {
        0x92, 0xde, 0xf0, 0x6b, 0x3c, 0x13, 0x0a, 0x59,
        0xdb, 0x54, 0xc7, 0x04, 0xf8, 0x18, 0x9d, 0x20,
        0x4a, 0x98, 0xfb, 0x2e, 0x67, 0xa8, 0x02, 0x4c,
        0x89, 0x12, 0x40, 0x9b, 0x17, 0xb5, 0x7e, 0x41,
    };
    const CK_BYTE ETALON[] = {
        0x15, 0x4E, 0x72, 0x10, 0x20, 0x30, 0xC5, 0xBB,
    };
    CK_BYTE value[sizeof(ETALON)];
    CK_ULONG valueLength = sizeof(value);
    CK_OBJECT_HANDLE secretKey = CK_INVALID_HANDLE;

    rv    =    pF->C_CreateObject(hSession,    secretKeyTemplate,
sizeof(secretKeyTemplate) / sizeof(CK_ATTRIBUTE), &secretKey);
    assert(rv == CKR_OK);

    rv = pF->C_SignInit(hSession, &mechanism, secretKey);
    assert(rv == CKR_OK);
    rv = pF->C_Sign(hSession, testData, sizeof(testData), value,
&valueLength);
    assert(rv == CKR_OK);
    assert(valueLength == sizeof(value));

    rv = pF->C_DestroyObject(hSession, secretKey);
    assert(rv == CKR_OK);
```

```
    return      memcmp(value,      ETALON,      valueLength)      ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.11 (справочное). Пример использования механизма CKM_MAGMA_KEXP_15_WRAP.

```
CK_RV         sample_kexp15_magma(CK_FUNCTION_LIST_PTR         pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE kekKeyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE cekKeyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE unwrappedKeyHandle = CK_INVALID_HANDLE;

    CK_BYTE iv[] = {
         0x67, 0xBE, 0xD6, 0x54,
    };
    CK_MECHANISM  kexpMechanism  =  {  CKM_MAGMA_KEXP_15_WRAP,  iv,
sizeof(iv) };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_MAGMA;
    CK_KEY_TYPE twinKeyType = CKK_MAGMA_TWIN_KEY;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;

    CK_BYTE cekKeyValue[] = {
        0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF,
        0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
        0xFE, 0xDC, 0xBA, 0x98, 0x76, 0x54, 0x32, 0x10,
        0x01, 0x23, 0x45, 0x67, 0x89, 0xAB, 0xCD, 0xEF,
    };
    CK_ATTRIBUTE cekKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_DECRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,       cekKeyValue,   sizeof(cekKeyValue)},
    };

    CK_BYTE kekKeyValue[] = {
         0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
         0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
         0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
         0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
         0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
         0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
         0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
```

```
    };
    CK_ATTRIBUTE kekKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &twinKeyType,  sizeof(twinKeyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_WRAP,        &bTrue,        sizeof(bTrue)},
         {CKA_UNWRAP,      &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,       kekKeyValue,   sizeof(kekKeyValue)},
    };

    CK_ATTRIBUTE unwrappedKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_EXTRACTABLE, &bTrue,        sizeof(bTrue)},
         {CKA_SENSITIVE,   &bFalse,       sizeof(bFalse)},
    };

    CK_BYTE ETALON[] = {
         0xCF, 0xD5, 0xA1, 0x2D, 0x5B, 0x81, 0xB6, 0xE1,
         0xE9, 0x9C, 0x91, 0x6D, 0x07, 0x90, 0x0C, 0x6A,
         0xC1, 0x27, 0x03, 0xFB, 0x3A, 0xBD, 0xED, 0x55,
         0x56, 0x7B, 0xF3, 0x74, 0x2C, 0x89, 0x9C, 0x75,
         0x5D, 0xAF, 0xE7, 0xB4, 0x2E, 0x3A, 0x8B, 0xD9,
    };

    rv     =     pF->C_CreateObject(hSession,     cekKeyTemplate,
sizeof(cekKeyTemplate) / sizeof(CK_ATTRIBUTE), &cekKeyHandle);
    assert(rv == CKR_OK);
    rv     =     pF->C_CreateObject(hSession,     kekKeyTemplate,
sizeof(kekKeyTemplate) / sizeof(CK_ATTRIBUTE), &kekKeyHandle);
    assert(rv == CKR_OK);

    CK_BYTE value[sizeof(ETALON)];
    CK_ULONG valueLength = sizeof(value);
    rv  =  pF->C_WrapKey(hSession,  &kexpMechanism,  kekKeyHandle,
cekKeyHandle, NULL, &valueLength);
    assert(rv == CKR_OK);
    assert(valueLength == sizeof(ETALON));
    rv  =  pF->C_WrapKey(hSession,  &kexpMechanism,  kekKeyHandle,
cekKeyHandle, value, &valueLength);
    assert(rv == CKR_OK);
    assert(valueLength == sizeof(ETALON));
    assert(memcmp(value, ETALON, valueLength) == 0);

    rv = pF->C_UnwrapKey(hSession, &kexpMechanism, kekKeyHandle,
ETALON, sizeof(ETALON),
        unwrappedKeyTemplate,   sizeof(unwrappedKeyTemplate)   /
sizeof(CK_ATTRIBUTE), &unwrappedKeyHandle);
    assert(rv == CKR_OK);
```

```
    CK_BYTE unwrappedKeyValue[sizeof(cekKeyValue)];
    CK_ATTRIBUTE keyAttribute = { CKA_VALUE, unwrappedKeyValue,
sizeof(unwrappedKeyValue) };
    rv = pF->C_GetAttributeValue(hSession, unwrappedKeyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);
    assert(keyAttribute.ulValueLen == sizeof(cekKeyValue));

    rv = pF->C_DestroyObject(hSession, unwrappedKeyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, kekKeyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, cekKeyHandle);
    assert(rv == CKR_OK);

    return         memcmp(unwrappedKeyValue,         cekKeyValue,
sizeof(cekKeyValue)) ? CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.12 (справочное). Пример использования механизма CKM_MAGMA_MGM.

```
CK_RV sample_magma_mgm(CK_FUNCTION_LIST_PTR pF, CK_SESSION_HANDLE
hSession)
{
    CK_RV rv = CKR_OK;
    CK_BYTE pIv[] = {
            0x12, 0xDE, 0xF0, 0x6B, 0x3C, 0x13, 0x0A, 0x59,
    };
    CK_BYTE addData[] = {
            0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
            0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
            0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
            0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
            0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
            0xEA,
    };
    CK_BYTE plainText[] = {
            0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
            0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x00,
            0x88, 0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A,
            0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77,
            0x99, 0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00,
            0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
            0xAA, 0xBB, 0xCC, 0xEE, 0xFF, 0x0A, 0x00, 0x11,
            0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, 0x99,
            0xAA, 0xBB, 0xCC,
    };
    CK_BYTE keyValue[] = {
            0xFF, 0xEE, 0xDD, 0xCC, 0xBB, 0xAA, 0x99, 0x88,
            0x77, 0x66, 0x55, 0x44, 0x33, 0x22, 0x11, 0x00,
```

```
            0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,
            0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF,
    };
    CK_BYTE etalonText[] = {
            0xC7, 0x95, 0x06, 0x6C, 0x5F, 0x9E, 0xA0, 0x3B,
            0x85, 0x11, 0x33, 0x42, 0x45, 0x91, 0x85, 0xAE,
            0x1F, 0x2E, 0x00, 0xD6, 0xBF, 0x2B, 0x78, 0x5D,
            0x94, 0x04, 0x70, 0xB8, 0xBB, 0x9C, 0x8E, 0x7D,
            0x9A, 0x5D, 0xD3, 0x73, 0x1F, 0x7D, 0xDC, 0x70,
            0xEC, 0x27, 0xCB, 0x0A, 0xCE, 0x6F, 0xA5, 0x76,
            0x70, 0xF6, 0x5C, 0x64, 0x6A, 0xBB, 0x75, 0xD5,
            0x47, 0xAA, 0x37, 0xC3, 0xBC, 0xB5, 0xC3, 0x4E,
            0x03, 0xBB, 0x9C,
    };
    CK_BYTE etalonTag[] = {
            0xA7, 0x92, 0x80, 0x69, 0xAA, 0x10, 0xFD, 0x10,
    };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_MAGMA;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
    CK_ATTRIBUTE secretKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_DECRYPT,     &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,       keyValue,      sizeof(keyValue)},
    };
    CK_ULONG secretKeyTemplateSize = sizeof(secretKeyTemplate) /
sizeof(*secretKeyTemplate);

    byte actualValue[sizeof(plainText) + sizeof(etalonTag)];
    byte decrypted[sizeof(plainText)];

    CK_GCM_PARAMS mechanismParam = {
         pIv, sizeof(pIv), sizeof(pIv) * 8,
         addData, sizeof(addData),
         sizeof(etalonTag) * 8 };
    CK_MECHANISM mechanism = { CKM_MAGMA_MGM, &mechanismParam,
sizeof(mechanismParam) };

    CK_OBJECT_HANDLE secretKey = CK_INVALID_HANDLE;
    rv    =    pF->C_CreateObject(hSession,    secretKeyTemplate,
secretKeyTemplateSize, &secretKey);
    assert(rv == CKR_OK);

    rv = pF->C_EncryptInit(hSession, &mechanism, secretKey);
    assert(rv == CKR_OK);
    CK_ULONG result = sizeof(actualValue);
```

```
    rv = pF->C_Encrypt(hSession, plainText, sizeof(plainText),
actualValue, &result);
    assert(rv == CKR_OK);
    assert(result == sizeof(actualValue));

    rv = pF->C_DecryptInit(hSession, &mechanism, secretKey);
    assert(rv == CKR_OK);
    result = sizeof(decrypted);
    rv       =       pF->C_Decrypt(hSession,       actualValue,
sizeof(actualValue), decrypted, &result);
    assert(rv == CKR_OK);
    assert(result == sizeof(decrypted));

    rv = pF->C_DestroyObject(hSession, secretKey);
    assert(rv == CKR_OK);

    assert(memcmp(etalonText, actualValue, sizeof(etalonText)) ==
0);
    assert(memcmp(plainText, decrypted, sizeof(plainText)) == 0);
    assert(memcmp(etalonTag,  actualValue  +  sizeof(plainText),
sizeof(etalonTag)) == 0);
    return CKR_OK;
}
```

Приложение 2.13 (справочное). Пример использования механизма CKM_KDF_HMAC3411_2012_256.

```
CK_RV sample_kdf_hmac(CK_FUNCTION_LIST_PTR pF, CK_SESSION_HANDLE
hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE derivedKeyHandle = CK_INVALID_HANDLE;

    CK_BYTE kdfHmacParams[] = {
         0x01,
         // label
         0x26, 0xbd, 0xb8, 0x78,
         0x00,
         // seed
         0xaf, 0x21, 0x43, 0x41, 0x45, 0x65, 0x63, 0x78,
         0x01, 0x00,
    };
    CK_MECHANISM deriveMechanism = { CKM_KDF_HMAC3411_2012_256,
kdfHmacParams, sizeof(kdfHmacParams) };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_MAGMA;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;

    CK_BYTE keyValue[] = {
         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
```

```
        0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
        0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
        0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
    };
    CK_ATTRIBUTE keyTemplate[] = {
        {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
        {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
        {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
        {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
        {CKA_DERIVE,      &bTrue,        sizeof(bTrue)},
        {CKA_VALUE,       keyValue,      sizeof(keyValue)},
    };

    CK_BYTE ETALON[] = {
        0xa1, 0xaa, 0x5f, 0x7d, 0xe4, 0x02, 0xd7, 0xb3,
        0xd3, 0x23, 0xf2, 0x99, 0x1c, 0x8d, 0x45, 0x34,
        0x01, 0x31, 0x37, 0x01, 0x0a, 0x83, 0x75, 0x4f,
        0xd0, 0xaf, 0x6d, 0x7c, 0xd4, 0x92, 0x2e, 0xd9,
    };

    CK_ATTRIBUTE derivedKeyTemplate[] = {
        {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
        {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
        {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
        {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
        {CKA_EXTRACTABLE, &bTrue,        sizeof(bTrue)},
        {CKA_SENSITIVE,   &bFalse,       sizeof(bFalse)},
    };

    rv      =      pF->C_CreateObject(hSession,      keyTemplate,
sizeof(keyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DeriveKey(hSession, &deriveMechanism, keyHandle,
derivedKeyTemplate,         sizeof(derivedKeyTemplate)         /
sizeof(CK_ATTRIBUTE), &derivedKeyHandle);
    assert(rv == CKR_OK);

    CK_BYTE value[sizeof(ETALON)];
    CK_ATTRIBUTE keyAttribute = { CKA_VALUE, value, sizeof(value)
};
    rv   =   pF->C_GetAttributeValue(hSession,   derivedKeyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);
    assert(keyAttribute.ulValueLen == sizeof(ETALON));

    rv = pF->C_DestroyObject(hSession, derivedKeyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, keyHandle);
    assert(rv == CKR_OK);
```

```
    return     memcmp(value,     ETALON,     sizeof(ETALON))     ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.14 (справочное). Пример использования механизма CKM_CONCATENATE_BASE_AND_KEY.

```
CK_RV         sample_concatenate(CK_FUNCTION_LIST_PTR         pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE macKeyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE encKeyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE twinKeyHandle = CK_INVALID_HANDLE;

    CK_MECHANISM         concatenateMechanism         =         {
CKM_CONCATENATE_BASE_AND_KEY, &encKeyHandle, sizeof(encKeyHandle)
};

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_MAGMA;
    CK_KEY_TYPE twinKeyType = CKK_MAGMA_TWIN_KEY;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;

    CK_BYTE macKeyValue[] = {
         0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
         0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
         0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
         0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
    };
    CK_BYTE encKeyValue[] = {
         0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27,
         0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D, 0x2E, 0x2F,
         0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
         0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    };

    CK_ATTRIBUTE macKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
         {CKA_SIGN,        &bTrue,        sizeof(bTrue)},
         {CKA_VERIFY,      &bTrue,        sizeof(bTrue)},
         {CKA_VALUE,       macKeyValue,   sizeof(macKeyValue)},
    };

    CK_ATTRIBUTE encKeyTemplate[] = {
         {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
         {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
         {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
         {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
```

```
        {CKA_ENCRYPT,     &bTrue,        sizeof(bTrue)},
        {CKA_DECRYPT,     &bTrue,        sizeof(bTrue)},
        {CKA_VALUE,       encKeyValue,   sizeof(encKeyValue)},
    };

    CK_ATTRIBUTE twinKeyTemplate[] = {
        {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
        {CKA_KEY_TYPE,    &twinKeyType,  sizeof(twinKeyType)},
        {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
        {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
        {CKA_WRAP,        &bTrue,        sizeof(bTrue)},
        {CKA_UNWRAP,      &bTrue,        sizeof(bTrue)},
        {CKA_EXTRACTABLE, &bTrue,        sizeof(bTrue)},
        {CKA_SENSITIVE,   &bFalse,       sizeof(bFalse)},
    };

    rv     =     pF->C_CreateObject(hSession,     macKeyTemplate,
sizeof(macKeyTemplate) / sizeof(CK_ATTRIBUTE), &macKeyHandle);
    assert(rv == CKR_OK);

    rv     =     pF->C_CreateObject(hSession,     encKeyTemplate,
sizeof(encKeyTemplate) / sizeof(CK_ATTRIBUTE), &encKeyHandle);
    assert(rv == CKR_OK);

    rv    =    pF->C_DeriveKey(hSession,    &concatenateMechanism,
macKeyHandle,    twinKeyTemplate,    sizeof(twinKeyTemplate)    /
sizeof(CK_ATTRIBUTE), &twinKeyHandle);
    assert(rv == CKR_OK);

    CK_BYTE           twinKeyValue[sizeof(macKeyValue)           +
sizeof(encKeyValue)];
    CK_ATTRIBUTE  keyAttribute  =  {  CKA_VALUE,  twinKeyValue,
sizeof(twinKeyValue) };
    rv    =    pF->C_GetAttributeValue(hSession,    twinKeyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);
    assert(keyAttribute.ulValueLen == sizeof(twinKeyValue));

    rv = pF->C_DestroyObject(hSession, twinKeyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, encKeyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, macKeyHandle);
    assert(rv == CKR_OK);

    return           (memcmp(twinKeyValue,           macKeyValue,
sizeof(macKeyValue)) &&
        memcmp(twinKeyValue  +  sizeof(macKeyValue),  encKeyValue,
sizeof(encKeyValue))) ? CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 2.15 (справочное). Пример использования механизма CKM_KDF_TREE_GOSTR3411_2012_256.

```
CK_RV sample_kdf_tree(CK_FUNCTION_LIST_PTR pF, CK_SESSION_HANDLE
hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE derivedKeyHandle = CK_INVALID_HANDLE;

    CK_BYTE label[] = { 0x26, 0xbd, 0xb8, 0x78, };
    CK_BYTE seed[] = { 0xaf, 0x21, 0x43, 0x41, 0x45, 0x65, 0x63,
0x78, };
    CK_KDF_TREE_GOST_PARAMS deriveParams = {
        sizeof(label), label,
        sizeof(seed), seed,
        1,
        64,
        32
    };
    CK_MECHANISM          deriveMechanism          =          {
CKM_KDF_TREE_GOSTR3411_2012_256,                   &deriveParams,
sizeof(deriveParams) };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_KUZNECHIK;
    CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;

    CK_BYTE keyValue[] = {
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f,
        0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
        0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f,
    };
    CK_ATTRIBUTE keyTemplate[] = {
        {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
        {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
        {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
        {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
        {CKA_DERIVE,      &bTrue,        sizeof(bTrue)},
        {CKA_VALUE,       keyValue,      sizeof(keyValue)},
    };

    CK_BYTE ETALON[] = {
        0x07, 0x4c, 0x93, 0x30, 0x59, 0x9d, 0x7f, 0x8d,
        0x71, 0x2f, 0xca, 0x54, 0x39, 0x2f, 0x4d, 0xdd,
        0xe9, 0x37, 0x51, 0x20, 0x6b, 0x35, 0x84, 0xc8,
        0xf4, 0x3f, 0x9e, 0x6d, 0xc5, 0x15, 0x31, 0xf9,
    };

    CK_ATTRIBUTE derivedKeyTemplate[] = {
        {CKA_CLASS,       &keyClass,     sizeof(keyClass)},
```

```
        {CKA_KEY_TYPE,    &keyType,      sizeof(keyType)},
        {CKA_TOKEN,       &bFalse,       sizeof(bFalse)},
        {CKA_PRIVATE,     &bTrue,        sizeof(bTrue)},
        {CKA_EXTRACTABLE, &bTrue,        sizeof(bTrue)},
        {CKA_SENSITIVE,   &bFalse,       sizeof(bFalse)},
    };

    rv      =      pF->C_CreateObject(hSession,      keyTemplate,
sizeof(keyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DeriveKey(hSession, &deriveMechanism, keyHandle,
derivedKeyTemplate,          sizeof(derivedKeyTemplate)          /
sizeof(CK_ATTRIBUTE), &derivedKeyHandle);
    assert(rv == CKR_OK);

    CK_BYTE value[sizeof(ETALON)];
    CK_ATTRIBUTE keyAttribute = { CKA_VALUE, value, sizeof(value)
};
    rv   =   pF->C_GetAttributeValue(hSession,   derivedKeyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);
    assert(keyAttribute.ulValueLen == sizeof(ETALON));

    rv = pF->C_DestroyObject(hSession, derivedKeyHandle);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, keyHandle);
    assert(rv == CKR_OK);

    return     memcmp(value,     ETALON,     sizeof(ETALON))     ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

3. **Приложение 3(справочное). Контрольные примеры использования ГОСТ Р 34.10-2012, ГОСТ Р 34.11-2012**

Приложение 3.1 (справочное). Пример использования объектов типа CKO_DOMAIN_PARAMETERS для определения возможности использования параметров эллиптических кривых.

```
CK_RV        sample_domain_param(CK_FUNCTION_LIST_PTR        pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv = CKR_OK;
     CK_OBJECT_HANDLE objectList[10] = { 0 };
     CK_ULONG     objectCount     =     sizeof(objectList)     /
sizeof(*objectList);

     CK_OBJECT_CLASS domainParamsClass = CKO_DOMAIN_PARAMETERS;
```

```
     CK_KEY_TYPE key_type_gost3410_256 = CKK_GOSTR3410;
     CK_KEY_TYPE key_type_gost3410_512 = CKK_GOSTR3410_512;
     CK_ATTRIBUTE domainParams_256[] =
     {
          {   CKA_CLASS,                      &domainParamsClass,
sizeof(domainParamsClass) },
          {   CKA_KEY_TYPE,              &key_type_gost3410_256,
sizeof(key_type_gost3410_256) },
     };
     CK_ATTRIBUTE domainParams_512[] =
     {
          {   CKA_CLASS,                      &domainParamsClass,
sizeof(domainParamsClass) },
          {   CKA_KEY_TYPE,              &key_type_gost3410_512,
sizeof(key_type_gost3410_512) },
     };
     const CK_BYTE default_param_256[] = {0x06, 0x07, 0x2a, 0x85,
0x03, 0x02, 0x02, 0x23, 0x01}; // "1.2.643.2.2.35.1"
     const CK_BYTE default_param_512[] = {0x06, 0x09, 0x2a, 0x85,
0x03,    0x07,    0x01,    0x02,    0x01,    0x02,    0x01};    //
"1.2.643.7.1.2.1.2.1"
     CK_BYTE domainParamValue[20];
     CK_ATTRIBUTE  attr  =  {  CKA_OBJECT_ID,  domainParamValue,
sizeof(domainParamValue) };

     rv  =  pF->C_FindObjectsInit(  hSession,  domainParams_256,
sizeof( domainParams_256 ) / sizeof( CK_ATTRIBUTE ) );
     assert(rv == CKR_OK);
     rv     =     pF->C_FindObjects(     hSession,     objectList,
sizeof(objectList) / sizeof(*objectList), &objectCount );
     assert(rv == CKR_OK);
     rv = pF->C_FindObjectsFinal( hSession );
     assert(rv == CKR_OK);
     for ( CK_ULONG i = 0; i < objectCount; i++ )
     {
          attr.ulValueLen = sizeof(domainParamValue);
          rv = pF->C_GetAttributeValue( hSession, objectList[i],
&attr, 1 );
          assert(rv == CKR_OK);
          if   (   memcmp(   domainParamValue,   default_param_256,
attr.ulValueLen ) == 0) {
               printf("default       256-bit       param       set
\"1.2.643.2.2.35.1\" supported.\n");
               break;
          }
     }

     rv  =  pF->C_FindObjectsInit(  hSession,  domainParams_512,
sizeof( domainParams_512 ) / sizeof( CK_ATTRIBUTE ) );
     assert(rv == CKR_OK);
     rv     =     pF->C_FindObjects(     hSession,     objectList,
sizeof(objectList) / sizeof(*objectList), &objectCount );
     assert(rv == CKR_OK);
```

```
     rv = pF->C_FindObjectsFinal( hSession );
     assert(rv == CKR_OK);
     for ( CK_ULONG i = 0; i < objectCount; i++ )
     {
          attr.ulValueLen = sizeof(domainParamValue);
          rv = pF->C_GetAttributeValue( hSession, objectList[i],
&attr, 1 );
          assert(rv == CKR_OK);
          if   (   memcmp(   domainParamValue,   default_param_512,
attr.ulValueLen ) == 0) {
               printf("default      512-bit      param      set
\"1.2.643.7.1.2.1.2.1\" supported.\n");
               break;
          }
     }
     return rv;
}
```

Приложение 3.2 (справочное). Пример использования механизма CKM_GOSTR3411_2012_512.

```
CK_RV     sample_digest_stribog_512(CK_FUNCTION_LIST_PTR     pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv;
     CK_BYTE pDigest[64];
     CK_ULONG ulDigestLen = sizeof(pDigest);
     CK_MECHANISM   digestMechanism   =   {   CKM_GOSTR3411_12_512,
NULL_PTR, 0 };
     // data from GOST R 3411-2012
     CK_BYTE                          data[]                          =
"012345678901234567890123456789012345678901234567890123456789012
";
     const CK_BYTE ETALON[] = {
          0x1B, 0x54, 0xD0, 0x1A, 0x4A, 0xF5, 0xB9, 0xD5,
          0xCC, 0x3D, 0x86, 0xD6, 0x8D, 0x28, 0x54, 0x62,
          0xB1, 0x9A, 0xBC, 0x24, 0x75, 0x22, 0x2F, 0x35,
          0xC0, 0x85, 0x12, 0x2B, 0xE4, 0xBA, 0x1F, 0xFA,
          0x00, 0xAD, 0x30, 0xF8, 0x76, 0x7B, 0x3A, 0x82,
          0x38, 0x4C, 0x65, 0x74, 0xF0, 0x24, 0xC3, 0x11,
          0xE2, 0xA4, 0x81, 0x33, 0x2B, 0x08, 0xEF, 0x7F,
          0x41, 0x79, 0x78, 0x91, 0xC1, 0x64, 0x6F, 0x48,
     };

     rv = pF->C_DigestInit(hSession, &digestMechanism);
     assert(rv == CKR_OK);
     rv = pF->C_DigestUpdate(hSession, data, sizeof(data)-1);
     assert(rv == CKR_OK);
     rv = pF->C_DigestFinal(hSession, pDigest, &ulDigestLen);
     assert(rv == CKR_OK);

     return     memcmp(pDigest,     ETALON,     ulDigestLen)     ?
CKR_FUNCTION_FAILED : CKR_OK;
```

```
}
```

Приложение 3.3 (справочное). Пример использования механизма CKM_GOSTR3411_2012_256.

```
CK_RV     sample_digest_stribog_256(CK_FUNCTION_LIST_PTR     pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv;
     CK_BYTE pDigest[32];
     CK_ULONG ulDigestLen = sizeof(pDigest);
     CK_MECHANISM  digestMechanism  =  {  CKM_GOSTR3411_12_256,
NULL_PTR, 0 };
     // data from GOST R 3411-2012
     CK_BYTE                          data[]                          =
"012345678901234567890123456789012345678901234567890123456789012
";
     const CK_BYTE ETALON[] = {
          0x9D, 0x15, 0x1E, 0xEF, 0xD8, 0x59, 0x0B, 0x89,
          0xDA, 0xA6, 0xBA, 0x6C, 0xB7, 0x4A, 0xF9, 0x27,
          0x5D, 0xD0, 0x51, 0x02, 0x6B, 0xB1, 0x49, 0xA4,
          0x52, 0xFD, 0x84, 0xE5, 0xE5, 0x7B, 0x55, 0x00,
     };

     rv = pF->C_DigestInit(hSession, &digestMechanism);
     assert(rv == CKR_OK);
     rv = pF->C_DigestUpdate(hSession, data, sizeof(data)-1);
     assert(rv == CKR_OK);
     rv = pF->C_DigestFinal(hSession, pDigest, &ulDigestLen);
     assert(rv == CKR_OK);

     return     memcmp(pDigest,     ETALON,     ulDigestLen)     ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.4 (справочное). Пример использования механизма CKM_GOSTR3411_2012_512_HMAC.

```
CK_RV      sample_hmac_stribog_512(CK_FUNCTION_LIST_PTR      pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv;
     CK_MECHANISM  hmacMechanism  =  {  CKM_GOSTR3411_12_512_HMAC,
NULL_PTR, 0};

     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE keyType = CKK_GENERIC_SECRET;
     CK_OBJECT_CLASS keyObject = CKO_SECRET_KEY;
     CK_BYTE keyValue[] = {
          0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
          0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
```

```
        0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
        0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
    };
    CK_ATTRIBUTE secretKeyTemplate[] = {
        {CKA_CLASS,         &keyObject,  sizeof(keyObject)},
        {CKA_KEY_TYPE,      &keyType,    sizeof(keyType)},
        {CKA_TOKEN,         &bFalse,     sizeof(bTrue)},
        {CKA_SIGN,          &bTrue,      sizeof(bTrue)},
        {CKA_VERIFY,        &bTrue,      sizeof(bTrue)},
        {CKA_EXTRACTABLE,   &bTrue,      sizeof(bTrue)},
        {CKA_VALUE,         keyValue,    sizeof(keyValue)},
    };
    CK_BYTE testData[] = {
        0x01, 0x26, 0xBD, 0xB8,  0x78, 0x00, 0xAF, 0x21,
        0x43, 0x41, 0x45, 0x65,  0x63, 0x78, 0x01, 0x00,
    };
    CK_OBJECT_HANDLE secretKey = CK_INVALID_HANDLE;
    CK_BYTE hmacValue[64];
    CK_ULONG hmacValueLength = sizeof(hmacValue);
    const CK_BYTE ETALON[] = {
        0xA5, 0x9B, 0xAB, 0x22, 0xEC, 0xAE, 0x19, 0xC6,
        0x5F, 0xBD, 0xE6, 0xE5, 0xF4, 0xE9, 0xF5, 0xD8,
        0x54, 0x9D, 0x31, 0xF0, 0x37, 0xF9, 0xDF, 0x9B,
        0x90, 0x55, 0x00, 0xE1, 0x71, 0x92, 0x3A, 0x77,
        0x3D, 0x5F, 0x15, 0x30, 0xF2, 0xED, 0x7E, 0x96,
        0x4C, 0xB2, 0xEE, 0xDC, 0x29, 0xE9, 0xAD, 0x2F,
        0x3A, 0xFE, 0x93, 0xB2, 0x81, 0x4F, 0x79, 0xF5,
        0x00, 0x0F, 0xFC, 0x03, 0x66, 0xC2, 0x51, 0xE6,
    };

    rv    =    pF->C_CreateObject(hSession,    secretKeyTemplate,
sizeof(secretKeyTemplate) / sizeof(CK_ATTRIBUTE), &secretKey);
    assert(rv == CKR_OK);

    rv = pF->C_SignInit(hSession, &hmacMechanism, secretKey);
    assert(rv == CKR_OK);
    rv   =   pF->C_Sign(hSession,   testData,   sizeof(testData),
hmacValue, &hmacValueLength);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, secretKey);
    assert(rv == CKR_OK);

    return   memcmp(hmacValue,   ETALON,   hmacValueLength)   ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.5 (справочное). Пример использования механизма CKM_GOSTR3411_2012_256_HMAC.

```
CK_RV      sample_hmac_stribog_256(CK_FUNCTION_LIST_PTR      pF,
CK_SESSION_HANDLE hSession)
{
```

```
    CK_RV rv = CKR_OK;
    CK_MECHANISM hmacMechanism = { CKM_GOSTR3411_12_256_HMAC,
NULL_PTR, 0};

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_GENERIC_SECRET;
    CK_OBJECT_CLASS keyObject = CKO_SECRET_KEY;
    CK_BYTE keyValue[] = {
        0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
        0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
        0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
    };
    CK_ATTRIBUTE secretKeyTemplate[] = {
        {CKA_CLASS,         &keyObject,  sizeof(keyObject)},
        {CKA_KEY_TYPE,      &keyType,    sizeof(keyType)},
        {CKA_TOKEN,         &bFalse,     sizeof(bTrue)},
        {CKA_SIGN,          &bTrue,      sizeof(bTrue)},
        {CKA_VERIFY,        &bTrue,      sizeof(bTrue)},
        {CKA_EXTRACTABLE,   &bTrue,      sizeof(bTrue)},
        {CKA_VALUE,         keyValue,    sizeof(keyValue)},
    };

    CK_BYTE testData[] = {
        0x01, 0x26, 0xBD, 0xB8,  0x78, 0x00, 0xAF, 0x21,
        0x43, 0x41, 0x45, 0x65,  0x63, 0x78, 0x01, 0x00,
    };
    CK_OBJECT_HANDLE secretKey = CK_INVALID_HANDLE;
    CK_BYTE hmacValue[32];
    CK_ULONG hmacValueLength = sizeof(hmacValue);
    const CK_BYTE ETALON[] = {
        0xA1, 0xAA, 0x5F, 0x7D, 0xE4, 0x02, 0xD7, 0xB3,
        0xD3, 0x23, 0xF2, 0x99, 0x1C, 0x8D, 0x45, 0x34,
        0x01, 0x31, 0x37, 0x01, 0x0A, 0x83, 0x75, 0x4F,
        0xD0, 0xAF, 0x6D, 0x7C, 0xD4, 0x92, 0x2E, 0xD9,
    };

    rv    =    pF->C_CreateObject(hSession,    secretKeyTemplate,
sizeof(secretKeyTemplate) / sizeof(CK_ATTRIBUTE), &secretKey);
    assert(rv == CKR_OK);

    rv = pF->C_SignInit(hSession, &hmacMechanism, secretKey);
    assert(rv == CKR_OK);
    rv   =   pF->C_Sign(hSession,   testData,   sizeof(testData),
hmacValue, &hmacValueLength);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, secretKey);
    assert(rv == CKR_OK);

    return   memcmp(hmacValue,   ETALON,   hmacValueLength)   ?
CKR_FUNCTION_FAILED : CKR_OK;
```

```
}
```

Приложение 3.6 (справочное). Примеры использования механизма CKM_TLS_GOST_PRF_2012_256

```
CK_RV        sample_prf_2012_256(CK_FUNCTION_LIST_PTR        pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv;
     CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;
     CK_BYTE keyValue[] =
     {
          0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
          0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
          0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
          0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
     };
     CK_BYTE seed[] = {
          0x18, 0x47, 0x1D, 0x62, 0x2D, 0xC6, 0x55, 0xC4,
          0xD2, 0xD2, 0x26, 0x96, 0x91, 0xCA, 0x4A, 0x56,
          0x0B, 0x50, 0xAB, 0xA6, 0x63, 0x55, 0x3A, 0xF2,
          0x41, 0xF1, 0xAD, 0xA8, 0x82, 0xC9, 0xF2, 0x9A,
     };
     CK_BYTE label[] = {
          0x11, 0x22, 0x33, 0x44, 0x55,
     };
     const CK_BYTE ETALON[] = {
          0xFF, 0x09, 0x66, 0x4A, 0x44, 0x74, 0x58, 0x65,
          0x94, 0x4F, 0x83, 0x9E, 0xBB, 0x48, 0x96, 0x5F,
          0x15, 0x44, 0xFF, 0x1C, 0xC8, 0xE8, 0xF1, 0x6F,
          0x24, 0x7E, 0xE5, 0xF8, 0xA9, 0xEB, 0xE9, 0x7F,
          0xC4, 0xE3, 0xC7, 0x90, 0x0E, 0x46, 0xCA, 0xD3,
          0xDB, 0x6A, 0x01, 0x64, 0x30, 0x63, 0x04, 0x0E,
          0xC6, 0x7F, 0xC0, 0xFD, 0x5C, 0xD9, 0xF9, 0x04,
          0x65, 0x23, 0x52, 0x37, 0xBD, 0xFF, 0x2C, 0x02,
     };

     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE keyType = CKK_GOST28147;
     CK_OBJECT_CLASS keyObject = CKO_SECRET_KEY;
     CK_ATTRIBUTE secretKeyTemplate[] = {
          {CKA_CLASS,       &keyObject, sizeof(keyObject)},
          {CKA_KEY_TYPE,    &keyType,   sizeof(keyType)},
          {CKA_TOKEN,       &bFalse,    sizeof(bTrue)},
          {CKA_VALUE,       keyValue,   sizeof(keyValue)},
          {CKA_DERIVE,      &bTrue,     sizeof(bTrue)},
          {CKA_EXTRACTABLE, &bTrue,     sizeof(bTrue)},
     };
     CK_TLS_PRF_PARAMS prfParams = { 0 };
     CK_ULONG outputLen = sizeof( ETALON );
     CK_BYTE outputBuf[sizeof( ETALON )];
```

```
     CK_MECHANISM   prfMech   =   {   CKM_TLS_GOST_PRF_2012_256,
&prfParams, sizeof( prfParams ) };

     rv   =   pF->C_CreateObject(hSession,   secretKeyTemplate,
sizeof(secretKeyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
     assert(rv == CKR_OK);

     prfParams.pSeed = seed;
     prfParams.ulSeedLen = sizeof( seed );
     prfParams.pLabel = label;
     prfParams.ulLabelLen = sizeof( label );
     prfParams.pOutput = outputBuf;
     prfParams.pulOutputLen = &outputLen;
     rv = pF->C_DeriveKey( hSession, &prfMech, keyHandle, NULL, 0,
NULL );
     assert(rv   ==   CKR_OK   &&   *prfParams.pulOutputLen   ==
sizeof(ETALON));

     rv = pF->C_DestroyObject( hSession, keyHandle );
     assert(rv == CKR_OK);

     return memcmp(prfParams.pOutput, ETALON, sizeof(ETALON)) ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.7 (справочное). Примеры использования механизма CKM_TLS_GOST_PRF_2012_512.

```
CK_RV        sample_prf_2012_512(CK_FUNCTION_LIST_PTR        pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv;
     CK_OBJECT_HANDLE keyHandle = CK_INVALID_HANDLE;
     CK_BYTE keyValue[] =
     {
          0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
          0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F,
          0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
          0x18, 0x19, 0x1A, 0x1B, 0x1C, 0x1D, 0x1E, 0x1F,
     };
     CK_BYTE seed[] = {
          0x18, 0x47, 0x1D, 0x62, 0x2D, 0xC6, 0x55, 0xC4,
          0xD2, 0xD2, 0x26, 0x96, 0x91, 0xCA, 0x4A, 0x56,
          0x0B, 0x50, 0xAB, 0xA6, 0x63, 0x55, 0x3A, 0xF2,
          0x41, 0xF1, 0xAD, 0xA8, 0x82, 0xC9, 0xF2, 0x9A,
     };
     CK_BYTE label[] = {
          0x11, 0x22, 0x33, 0x44, 0x55,
     };
     const CK_BYTE ETALON[] = {
          0xF3, 0x51, 0x87, 0xA3, 0xDC, 0x96, 0x55, 0x11,
          0x3A, 0x0E, 0x84, 0xD0, 0x6F, 0xD7, 0x52, 0x6C,
          0x5F, 0xC1, 0xFB, 0xDE, 0xC1, 0xA0, 0xE4, 0x67,
```

```
            0x3D, 0xD6, 0xD7, 0x9D, 0x0B, 0x92, 0x0E, 0x65,
            0xAD, 0x1B, 0xC4, 0x7B, 0xB0, 0x83, 0xB3, 0x85,
            0x1C, 0xB7, 0xCD, 0x8E, 0x7E, 0x6A, 0x91, 0x1A,
            0x62, 0x6C, 0xF0, 0x2B, 0x29, 0xE9, 0xE4, 0xA5,
            0x8E, 0xD7, 0x66, 0xA4, 0x49, 0xA7, 0x29, 0x6D,
            0xE6, 0x1A, 0x7A, 0x26, 0xC4, 0xD1, 0xCA, 0xEE,
            0xCF, 0xD8, 0x0C, 0xCA, 0x65, 0xC7, 0x1F, 0x0F,
            0x88, 0xC1, 0xF8, 0x22, 0xC0, 0xE8, 0xC0, 0xAD,
            0x94, 0x9D, 0x03, 0xFE, 0xE1, 0x39, 0x57, 0x9F,
            0x72, 0xBA, 0x0C, 0x3D, 0x32, 0xC5, 0xF9, 0x54,
            0xF1, 0xCC, 0xCD, 0x54, 0x08, 0x1F, 0xC7, 0x44,
            0x02, 0x78, 0xCB, 0xA1, 0xFE, 0x7B, 0x7A, 0x17,
            0xA9, 0x86, 0xFD, 0xFF, 0x5B, 0xD1, 0x5D, 0x1F,
    };

    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE keyType = CKK_GENERIC_SECRET;
    CK_OBJECT_CLASS keyObject = CKO_SECRET_KEY;
    CK_ATTRIBUTE secretKeyTemplate[] = {
        {CKA_CLASS,       &keyObject, sizeof(keyObject)},
        {CKA_KEY_TYPE,    &keyType,   sizeof(keyType)},
        {CKA_TOKEN,       &bFalse,    sizeof(bTrue)},
        {CKA_VALUE,       keyValue,   sizeof(keyValue)},
        {CKA_DERIVE,      &bTrue,     sizeof(bTrue)},
        {CKA_EXTRACTABLE, &bTrue,     sizeof(bTrue)},
    };
    CK_TLS_PRF_PARAMS prfParams = { 0 };
    CK_ULONG outputLen = sizeof( ETALON );
    CK_BYTE outputBuf[sizeof( ETALON )];
    CK_MECHANISM   prfMech   =   {   CKM_TLS_GOST_PRF_2012_512,
&prfParams, sizeof( prfParams ) };

    rv    =    pF->C_CreateObject(hSession,    secretKeyTemplate,
sizeof(secretKeyTemplate) / sizeof(CK_ATTRIBUTE), &keyHandle);
    assert(rv == CKR_OK);

    prfParams.pSeed = seed;
    prfParams.ulSeedLen = sizeof( seed );
    prfParams.pLabel = label;
    prfParams.ulLabelLen = sizeof( label );
    prfParams.pOutput = outputBuf;
    prfParams.pulOutputLen = &outputLen;
    rv = pF->C_DeriveKey( hSession, &prfMech, keyHandle, NULL, 0,
NULL );
    assert(rv   ==   CKR_OK   &&   *prfParams.pulOutputLen   ==
sizeof(ETALON));

    rv = pF->C_DestroyObject( hSession, keyHandle );
    assert(rv == CKR_OK);

    return memcmp(prfParams.pOutput, ETALON, sizeof(ETALON)) ?
CKR_FUNCTION_FAILED : CKR_OK;
```

```
}
```

Приложение 3.8 (справочное). Примеры использования алгоритма хэширования ГОСТ Р 34.11-2012 в механизме генерации ключей PBKDF2.

```
CK_RV      sample_pbkdf_stribog_512(CK_FUNCTION_LIST_PTR      pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv = CKR_OK;
     const char salt[] = "saltSALTsalt";
     const char password[] = "password";
     CK_ULONG password_length = sizeof(password) - 1;

     CK_PKCS5_PBKD2_PARAMS2 params = {
          CKZ_SALT_SPECIFIED,
          (CK_BYTE_PTR)salt, sizeof(salt) - 1,
          2048,
          CKP_PKCS5_PBKD2_HMAC_GOSTR3411_2012_512,
          NULL, 0,
          (CK_UTF8CHAR_PTR)password, password_length,
     };
     CK_MECHANISM  pbkdfMechanism  =  {  CKM_PKCS5_PBKD2,  &params,
sizeof(CK_PKCS5_PBKD2_PARAMS2) };

     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE keyType = CKK_GOST28147;
     CK_OBJECT_CLASS keyObject = CKO_SECRET_KEY;
     CK_ATTRIBUTE secretKeyTemplate[] = {
          {CKA_CLASS,      &keyObject,  sizeof(keyObject)},
          {CKA_KEY_TYPE,   &keyType,    sizeof(keyType)},
          {CKA_TOKEN,      &bFalse,     sizeof(bTrue)},
          {CKA_ENCRYPT,    &bTrue,      sizeof(bTrue)},
          {CKA_DECRYPT,    &bTrue,      sizeof(bTrue)},
          {CKA_EXTRACTABLE,&bTrue,      sizeof(bTrue)},
     };

     CK_OBJECT_HANDLE secretKey;
     CK_BYTE keyValue[32];
     CK_ATTRIBUTE    keyAttribute    =    {CKA_VALUE,    keyValue,
sizeof(keyValue)};
     const CK_BYTE ETALON[] = {
          0x96, 0x85, 0x54, 0x56, 0xF3, 0x1E, 0x87, 0xD8,
          0xCA, 0x4F, 0x55, 0x62, 0x91, 0xDE, 0x76, 0x7C,
          0x97, 0xEF, 0x3F, 0x59, 0x7E, 0x65, 0xBA, 0x86,
          0x82, 0x70, 0xE9, 0x41, 0x24, 0xCF, 0x68, 0x24,
     };

     rv = pF->C_GenerateKey(hSession, &pbkdfMechanism,
          secretKeyTemplate,     sizeof(secretKeyTemplate)     /
sizeof(CK_ATTRIBUTE), &secretKey);
     assert(rv == CKR_OK);
```

```
     rv     =     pF->C_GetAttributeValue(hSession,     secretKey,
&keyAttribute, 1);
     assert(rv == CKR_OK);
     rv = pF->C_DestroyObject(hSession, secretKey);
     assert(rv == CKR_OK);

     return   memcmp(keyValue,   ETALON,   sizeof(keyValue))   ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.9 (справочное). Пример использования механизма CKM_GOSTR3410_512_KEY_PAIR_GEN для генерации ключевой пары.

```
CK_RV    sample_generate_key_pair_512(CK_FUNCTION_LIST_PTR    pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv = CKR_OK;
     CK_MECHANISM              asymKeyMechanism512              =
{CKM_GOSTR3410_512_KEY_PAIR_GEN, 0, 0};
     CK_OBJECT_HANDLE publicKey512 = CK_INVALID_HANDLE;
     CK_OBJECT_HANDLE privateKey512 = CK_INVALID_HANDLE;

     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE keyType2012 = CKK_GOSTR3410_512;
     CK_OBJECT_CLASS pubkeyClass = CKO_PUBLIC_KEY;
     CK_OBJECT_CLASS privkeyClass = CKO_PRIVATE_KEY;

     CK_ATTRIBUTE PublicKey512Template[] = {
          {CKA_CLASS,                                  &pubkeyClass,
sizeof(pubkeyClass) },
          {CKA_KEY_TYPE,                               &keyType2012,
sizeof(keyType2012) },
          {CKA_TOKEN,            &bFalse,          sizeof(bFalse)
},
          {CKA_PRIVATE,          &bFalse,          sizeof(bFalse)
},
          {CKA_VERIFY,           &bTrue,           sizeof(bTrue)
},
     };

     CK_ATTRIBUTE PrivateKey512Template[] = {
          {CKA_CLASS,                                  &privkeyClass,
sizeof(privkeyClass) },
          {CKA_KEY_TYPE,                               &keyType2012,
sizeof(keyType2012) },
          {CKA_TOKEN,            &bFalse,          sizeof(bFalse)
},
          {CKA_PRIVATE,          &bTrue,           sizeof(bTrue)
},
          {CKA_EXTRACTABLE,      &bTrue,           sizeof(bTrue)
},
```

```
          {CKA_SIGN,              &bTrue,           sizeof(bTrue)
},
     };
     CK_ATTRIBUTE keyAttribute = {CKA_VALUE, NULL, 0};

     rv = pF->C_GenerateKeyPair(hSession, &asymKeyMechanism512,
          PublicKey512Template,  sizeof(PublicKey512Template)  /
sizeof(CK_ATTRIBUTE),
          PrivateKey512Template, sizeof(PrivateKey512Template) /
sizeof(CK_ATTRIBUTE),
          &publicKey512, &privateKey512);
     assert(rv == CKR_OK);

     rv   =   pF->C_GetAttributeValue(hSession,   publicKey512,
&keyAttribute, 1);
     assert(rv == CKR_OK && keyAttribute.ulValueLen == 128);

     rv = pF->C_DestroyObject(hSession, publicKey512);
     assert(rv == CKR_OK);
     rv = pF->C_DestroyObject(hSession, privateKey512);
     assert(rv == CKR_OK);

     return CKR_OK;
}
```

Приложение 3.10 (справочное). Пример использования механизма CKM_GOSTR3410_PUBLIC_KEY_DERIVE для получения открытого ключа.

```
CK_RV        sample_public_derive(CK_FUNCTION_LIST_PTR        pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv = CKR_OK;
     CK_OBJECT_HANDLE publicKeyHandle = CK_INVALID_HANDLE;
     CK_OBJECT_HANDLE privateKeyHandle = CK_INVALID_HANDLE;
     CK_MECHANISM                deriveMechanism                =
{CKM_GOSTR3410_PUBLIC_KEY_DERIVE, 0, 0};

     CK_BYTE keyValue[] = {
          0xD9, 0x2D, 0x43, 0x1D, 0x20, 0x37, 0x5C, 0xD2,
          0xA5, 0x37, 0xCD, 0x64, 0x8E, 0x14, 0xB6, 0x0B,
          0x4C, 0x21, 0xA1, 0x5A, 0x57, 0x98, 0x61, 0xB7,
          0xBE, 0x41, 0x9B, 0x16, 0xED, 0x86, 0x18, 0x74,
     };
     CK_BYTE gost3410_defOid[] = { 0x06, 0x07, 0x2a, 0x85, 0x03,
0x02, 0x02, 0x23, 0x01 };
     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE keyType = CKK_GOSTR3410;
     CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
     CK_OBJECT_CLASS publicClass = CKO_PUBLIC_KEY;
     CK_ATTRIBUTE privateKeyTemplate[] = {
          {  CKA_CLASS,                          &privateClass,
sizeof(privateClass) },
```

```
          { CKA_KEY_TYPE,         &keyType,        sizeof(keyType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bTrue,          sizeof(bTrue)
},
          { CKA_VALUE,            keyValue,        sizeof(keyValue)
},
          { CKA_SIGN,              &bTrue,          sizeof(bTrue)
},
          { CKA_DERIVE,            &bTrue,          sizeof(bTrue)
},
          {         CKA_GOSTR3410_PARAMS,         gost3410_defOid,
sizeof(gost3410_defOid) },
     };

     CK_ATTRIBUTE publicKeyTemplate[] =
     {
          {  CKA_CLASS,                              &publicClass,
sizeof(publicClass)},
          { CKA_KEY_TYPE,         &keyType,        sizeof(keyType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bFalse,         sizeof(bFalse)
},
          { CKA_VERIFY,            &bTrue,          sizeof(bTrue)
},
     };
     const CK_BYTE ETALON[] = {
          0x03, 0x06, 0x54, 0xAC, 0xD1, 0x4A, 0xD8, 0x5D,
          0x6B, 0x24, 0x6E, 0xC4, 0xA1, 0x95, 0xB3, 0x34,
          0xEC, 0xFE, 0xF9, 0x3C, 0x1F, 0x22, 0xB6, 0x7C,
          0xF8, 0x1F, 0xF7, 0xD3, 0x5E, 0x8D, 0xD6, 0x18,
          0xE5, 0x38, 0xC3, 0xB3, 0x27, 0xE9, 0x3B, 0x13,
          0x66, 0x97, 0xED, 0x5C, 0x86, 0x17, 0x3B, 0x44,
          0x34, 0x1C, 0x5F, 0x5B, 0x97, 0x92, 0xE9, 0x53,
          0x62, 0x17, 0x0A, 0x99, 0x3D, 0x84, 0xA4, 0x72,
     };
     CK_BYTE publicKeyValue[64];
     CK_ATTRIBUTE  keyAttribute  =  {CKA_VALUE,  publicKeyValue,
sizeof(publicKeyValue)};

     rv  =  pF->C_CreateObject(  hSession,  privateKeyTemplate,
sizeof(  privateKeyTemplate  )  /  sizeof(  CK_ATTRIBUTE  ),
&privateKeyHandle );
     assert(rv == CKR_OK);

     rv   =   pF->C_DeriveKey(   hSession,   &deriveMechanism,
privateKeyHandle, publicKeyTemplate, sizeof( publicKeyTemplate )
/ sizeof( CK_ATTRIBUTE ), &publicKeyHandle );
     assert(rv == CKR_OK);
```

```
     rv   =   pF->C_GetAttributeValue(hSession,   publicKeyHandle,
&keyAttribute, 1);
     assert(rv == CKR_OK);
     rv = pF->C_DestroyObject(hSession, publicKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_DestroyObject(hSession, privateKeyHandle);
     assert(rv == CKR_OK);

     return            memcmp(publicKeyValue,            ETALON,
sizeof(publicKeyValue)) ? CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.11 (справочное). Пример использования механизмов для подписи и проверки по ГОСТ Р 34.10-2012.

```
CK_RV         sample_sign_verify(CK_FUNCTION_LIST_PTR         pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv = CKR_OK;
     CK_OBJECT_HANDLE publicKeyHandle = CK_INVALID_HANDLE;
     CK_OBJECT_HANDLE privateKeyHandle = CK_INVALID_HANDLE;

     CK_BYTE privateValue[] = {
          0xD9, 0x2D, 0x43, 0x1D, 0x20, 0x37, 0x5C, 0xD2,
          0xA5, 0x37, 0xCD, 0x64, 0x8E, 0x14, 0xB6, 0x0B,
          0x4C, 0x21, 0xA1, 0x5A, 0x57, 0x98, 0x61, 0xB7,
          0xBE, 0x41, 0x9B, 0x16, 0xED, 0x86, 0x18, 0x74,
     };
     CK_BYTE gost3410_defOid[] = { 0x06, 0x07, 0x2a, 0x85, 0x03,
0x02, 0x02, 0x23, 0x01 };
     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE keyType = CKK_GOSTR3410;
     CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
     CK_OBJECT_CLASS publicClass = CKO_PUBLIC_KEY;
     CK_ATTRIBUTE privateKeyTemplate[] = {
          {   CKA_CLASS,                              &privateClass,
sizeof(privateClass) },
          { CKA_KEY_TYPE,          &keyType,         sizeof(keyType)
},
          { CKA_TOKEN,              &bFalse,          sizeof(bFalse)
},
          { CKA_PRIVATE,            &bTrue,           sizeof(bTrue)
},
          {   CKA_VALUE,                              privateValue,
sizeof(privateValue) },
          { CKA_SIGN,               &bTrue,           sizeof(bTrue)
},
          { CKA_DERIVE,             &bTrue,           sizeof(bTrue)
},
          {         CKA_GOSTR3410_PARAMS,          gost3410_defOid,
sizeof(gost3410_defOid) },
     };
```

```
     CK_BYTE publicValue[] = {
          0x03, 0x06, 0x54, 0xAC, 0xD1, 0x4A, 0xD8, 0x5D,
          0x6B, 0x24, 0x6E, 0xC4, 0xA1, 0x95, 0xB3, 0x34,
          0xEC, 0xFE, 0xF9, 0x3C, 0x1F, 0x22, 0xB6, 0x7C,
          0xF8, 0x1F, 0xF7, 0xD3, 0x5E, 0x8D, 0xD6, 0x18,
          0xE5, 0x38, 0xC3, 0xB3, 0x27, 0xE9, 0x3B, 0x13,
          0x66, 0x97, 0xED, 0x5C, 0x86, 0x17, 0x3B, 0x44,
          0x34, 0x1C, 0x5F, 0x5B, 0x97, 0x92, 0xE9, 0x53,
          0x62, 0x17, 0x0A, 0x99, 0x3D, 0x84, 0xA4, 0x72,
     };
     CK_ATTRIBUTE publicKeyTemplate[] =
     {
          {  CKA_CLASS,                              &publicClass,
sizeof(publicClass)},
          { CKA_KEY_TYPE,         &keyType,        sizeof(keyType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bFalse,         sizeof(bFalse)
},
          {  CKA_VALUE,                               publicValue,
sizeof(publicValue) },
          { CKA_VERIFY,            &bTrue,          sizeof(bTrue)
},
          {        CKA_GOSTR3410_PARAMS,        gost3410_defOid,
sizeof(gost3410_defOid) },
     };
     CK_BYTE pangram[] = "The quick brown fox jumps over the lazy
dog";
     CK_BYTE pangramDigest[] = {
          0x3e, 0x7d, 0xea, 0x7f, 0x23, 0x84, 0xb6, 0xc5,
          0xa3, 0xd0, 0xe2, 0x4a, 0xaa, 0x29, 0xc0, 0x5e,
          0x89, 0xdd, 0xd7, 0x62, 0x14, 0x50, 0x30, 0xec,
          0x22, 0xc7, 0x1a, 0x6d, 0xb8, 0xb2, 0xc1, 0xf4,
     };

     CK_MECHANISM signOnlyMechanism = {CKM_GOSTR3410_256, 0, 0};
     CK_MECHANISM             signWithHashMechanism             =
{CKM_GOSTR3410_WITH_GOSTR3411_2012_256, 0, 0};
     CK_ULONG signatureLen = 0;
     CK_BYTE hashSignature[64];
     CK_BYTE dataSignature[64];
     CK_BYTE ETALON[] = {
          0x68, 0x13, 0x4d, 0x22, 0xa3, 0xf3, 0xb0, 0x70,
          0x7a, 0x85, 0xc9, 0xb8, 0x8f, 0xaf, 0x12, 0x9c,
          0x1b, 0x83, 0xca, 0x26, 0x31, 0x1c, 0x1f, 0x47,
          0xbd, 0x5f, 0xaa, 0x00, 0x13, 0x45, 0x45, 0x19,
          0xcf, 0x17, 0x36, 0x16, 0x8b, 0xa1, 0xb7, 0x01,
          0x48, 0xd8, 0x86, 0xf2, 0x67, 0x7c, 0xa4, 0xc6,
          0x8e, 0xd9, 0xf2, 0xbe, 0x42, 0x4b, 0x25, 0x08,
          0x40, 0x00, 0x08, 0x70, 0xd6, 0xd3, 0x98, 0xba,
     };
```

```
     rv = pF->C_CreateObject( hSession, privateKeyTemplate,
sizeof( privateKeyTemplate ) / sizeof( CK_ATTRIBUTE ),
&privateKeyHandle );
     assert(rv == CKR_OK);
     rv = pF->C_CreateObject( hSession, publicKeyTemplate, sizeof(
publicKeyTemplate ) / sizeof( CK_ATTRIBUTE ), &publicKeyHandle );
     assert(rv == CKR_OK);

     rv = pF->C_SignInit(hSession, &signOnlyMechanism,
privateKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Sign(hSession, pangramDigest,
sizeof(pangramDigest), NULL, &signatureLen);
     assert(rv == CKR_OK);
     assert(signatureLen == 64);
     rv = pF->C_Sign(hSession, pangramDigest,
sizeof(pangramDigest), hashSignature, &signatureLen);
     assert(rv == CKR_OK);

     rv = pF->C_SignInit(hSession, &signWithHashMechanism,
privateKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Sign(hSession, pangram, sizeof(pangram) - 1, NULL,
&signatureLen);
     assert(rv == CKR_OK);
     assert(signatureLen == 64);
     rv = pF->C_Sign(hSession, pangram, sizeof(pangram) - 1,
dataSignature, &signatureLen);
     assert(rv == CKR_OK);

     rv = pF->C_VerifyInit(hSession, &signOnlyMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Verify(hSession, pangramDigest,
sizeof(pangramDigest), hashSignature, signatureLen);
     assert(rv == CKR_OK);
     rv = pF->C_VerifyInit(hSession, &signOnlyMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Verify(hSession, pangramDigest,
sizeof(pangramDigest), dataSignature, signatureLen);
     assert(rv == CKR_OK);
     rv = pF->C_VerifyInit(hSession, &signOnlyMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Verify(hSession, pangramDigest,
sizeof(pangramDigest), ETALON, signatureLen);
     assert(rv == CKR_OK);

     rv = pF->C_VerifyInit(hSession, &signWithHashMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
```

```
     rv = pF->C_Verify(hSession, pangram, sizeof(pangram) - 1,
hashSignature, signatureLen);
     assert(rv == CKR_OK);
     rv = pF->C_VerifyInit(hSession, &signWithHashMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Verify(hSession, pangram, sizeof(pangram) - 1,
dataSignature, signatureLen);
     assert(rv == CKR_OK);
     rv = pF->C_VerifyInit(hSession, &signWithHashMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Verify(hSession, pangram, sizeof(pangram) - 1,
ETALON, signatureLen);
     assert(rv == CKR_OK);

     rv = pF->C_DestroyObject(hSession, publicKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_DestroyObject(hSession, privateKeyHandle);
     assert(rv == CKR_OK);

     return CKR_OK;
}
```

Приложение 3.12 (справочное). Пример использования механизмов для подписи и проверки по ГОСТ Р 34.10-2012.

```
CK_RV       sample_sign_verify_512(CK_FUNCTION_LIST_PTR       pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv = CKR_OK;
     CK_OBJECT_HANDLE publicKeyHandle = CK_INVALID_HANDLE;
     CK_OBJECT_HANDLE privateKeyHandle = CK_INVALID_HANDLE;

     CK_BYTE privateValue[] = {
          0xC2, 0x48, 0x02, 0x82, 0x70, 0xE0, 0xFF, 0x17,
          0xD4, 0xDD, 0x9D, 0xA7, 0x19, 0xE2, 0xBD, 0xB6,
          0xDF, 0x60, 0x17, 0x2B, 0xCB, 0xC1, 0x70, 0x9A,
          0xBC, 0x4B, 0xAA, 0x80, 0xD2, 0xB6, 0x56, 0x9B,
          0x69, 0xDC, 0xED, 0x7A, 0x02, 0x66, 0xAC, 0xE0,
          0xA2, 0x64, 0x2C, 0xB4, 0x3A, 0x35, 0x87, 0x8F,
          0x82, 0x5F, 0x30, 0x2F, 0x14, 0x63, 0xDE, 0xC0,
          0xB7, 0x41, 0x33, 0xAF, 0x55, 0x81, 0x65, 0x40,
     };
     CK_BYTE gost3410_defOid[] = { 0x06, 0x09, 0x2a, 0x85, 0x03,
0x07, 0x01, 0x02, 0x01, 0x02, 0x01 }; //1.2.643.7.1.2.1.2.1
     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE keyType = CKK_GOSTR3410_512;
     CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
     CK_OBJECT_CLASS publicClass = CKO_PUBLIC_KEY;
     CK_ATTRIBUTE privateKeyTemplate[] = {
```

```
          {  CKA_CLASS,                              &privateClass,
sizeof(privateClass) },
          { CKA_KEY_TYPE,         &keyType,        sizeof(keyType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bTrue,           sizeof(bTrue)
},
          {  CKA_VALUE,                              privateValue,
sizeof(privateValue) },
          { CKA_SIGN,              &bTrue,           sizeof(bTrue)
},
          { CKA_DERIVE,            &bTrue,           sizeof(bTrue)
},
          {         CKA_GOSTR3410_PARAMS,         gost3410_defOid,
sizeof(gost3410_defOid) },
     };

     CK_BYTE publicValue[] = {
          0xf1, 0xa7, 0x56, 0x64, 0xfd, 0xa4, 0x27, 0x64,
          0xe4, 0x9f, 0x0d, 0x73, 0xae, 0x95, 0x56, 0x65,
          0xba, 0x6c, 0x27, 0x97, 0x2f, 0x8e, 0x79, 0x30,
          0xe6, 0x77, 0x7f, 0xb8, 0xd1, 0xf7, 0xa5, 0xc8,
          0x97, 0x4c, 0x5f, 0x15, 0xa5, 0x75, 0x94, 0x84,
          0x53, 0x9c, 0x21, 0xea, 0x8b, 0x15, 0xba, 0x29,
          0x02, 0x82, 0x54, 0x30, 0x72, 0xdf, 0x48, 0xea,
          0x62, 0x32, 0x41, 0xf0, 0x21, 0xb5, 0x0e, 0xab,
          0xb3, 0x34, 0x59, 0x11, 0x82, 0x83, 0x0c, 0xb6,
          0x7c, 0x5a, 0x33, 0x9d, 0x53, 0x78, 0xf3, 0x42,
          0x51, 0x8b, 0xeb, 0xcb, 0xa9, 0x49, 0x1e, 0xb6,
          0xcf, 0xb9, 0x75, 0x51, 0x7f, 0x17, 0x4a, 0xab,
          0x5b, 0x5d, 0x3b, 0xc1, 0x03, 0x61, 0x85, 0xa9,
          0x25, 0x26, 0x9d, 0xca, 0x4a, 0xe5, 0xb3, 0xe9,
          0x1e, 0x17, 0x3d, 0xda, 0xb1, 0x64, 0xfa, 0x98,
          0x6d, 0x17, 0xd4, 0x1c, 0x3f, 0x33, 0x7b, 0x4c,
     };
     CK_ATTRIBUTE publicKeyTemplate[] =
     {
          {  CKA_CLASS,                               &publicClass,
sizeof(publicClass)},
          { CKA_KEY_TYPE,         &keyType,        sizeof(keyType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bFalse,         sizeof(bFalse)
},
          {  CKA_VALUE,                               publicValue,
sizeof(publicValue) },
          { CKA_VERIFY,            &bTrue,           sizeof(bTrue)
},
          {         CKA_GOSTR3410_PARAMS,         gost3410_defOid,
sizeof(gost3410_defOid) },
     };
```

```
     CK_BYTE pangram[] = "The quick brown fox jumps over the lazy
dog";
     CK_BYTE pangramDigest[] = {
          0xd2, 0xb7, 0x93, 0xa0, 0xbb, 0x6c, 0xb5, 0x90,
          0x48, 0x28, 0xb5, 0xb6, 0xdc, 0xfb, 0x44, 0x3b,
          0xb8, 0xf3, 0x3e, 0xfc, 0x06, 0xad, 0x09, 0x36,
          0x88, 0x78, 0xae, 0x4c, 0xdc, 0x82, 0x45, 0xb9,
          0x7e, 0x60, 0x80, 0x24, 0x69, 0xbe, 0xd1, 0xe7,
          0xc2, 0x1a, 0x64, 0xff, 0x0b, 0x17, 0x9a, 0x6a,
          0x1e, 0x0b, 0xb7, 0x4d, 0x92, 0x96, 0x54, 0x50,
          0xa0, 0xad, 0xab, 0x69, 0x16, 0x2c, 0x00, 0xfe,
     };

     CK_MECHANISM signOnlyMechanism = {CKM_GOSTR3410_512, 0, 0};
     CK_MECHANISM             signWithHashMechanism             =
{CKM_GOSTR3410_WITH_GOSTR3411_2012_512, 0, 0};
     CK_ULONG signatureLen = 0;
     CK_BYTE hashSignature[128];
     CK_BYTE dataSignature[128];
     CK_BYTE ETALON[] = {
          0x38, 0x11, 0x87, 0x26, 0xe0, 0x05, 0xa3, 0x86,
          0x7e, 0xe5, 0xd4, 0xa8, 0x89, 0x41, 0x8d, 0x41,
          0x17, 0x66, 0x1b, 0x4d, 0xdc, 0x15, 0x95, 0x89,
          0xb1, 0x45, 0xcf, 0x42, 0x49, 0x1c, 0xb9, 0xe5,
          0xf6, 0x30, 0x69, 0x13, 0x55, 0x9b, 0x10, 0xd8,
          0xa9, 0x0d, 0xee, 0xd6, 0x55, 0xf2, 0xbb, 0xff,
          0x6c, 0xac, 0xa6, 0xcd, 0xea, 0xcc, 0x56, 0x67,
          0x03, 0x52, 0xeb, 0xf2, 0x70, 0xee, 0x12, 0xba,
          0x52, 0x42, 0x9f, 0x17, 0x3d, 0xd2, 0xd1, 0x02,
          0x98, 0x4c, 0x67, 0xce, 0xea, 0xcb, 0xf3, 0x98,
          0xcc, 0x17, 0x4f, 0x06, 0x7d, 0x4b, 0xeb, 0xf5,
          0xe5, 0xe5, 0xf8, 0x6b, 0x19, 0x36, 0x95, 0x25,
          0xb4, 0x2e, 0xca, 0x0a, 0xa8, 0xe3, 0x69, 0xa9,
          0xe7, 0xd3, 0x86, 0x21, 0x0c, 0x7a, 0x25, 0x42,
          0x2c, 0xff, 0x04, 0x3f, 0x9d, 0xe9, 0xe4, 0xef,
          0xfb, 0x33, 0x70, 0xa4, 0x3e, 0xa9, 0x17, 0x7c,
     };

     rv   =   pF->C_CreateObject(   hSession,   privateKeyTemplate,
sizeof(   privateKeyTemplate   )   /   sizeof(   CK_ATTRIBUTE   ),
&privateKeyHandle );
     assert(rv == CKR_OK);
     rv = pF->C_CreateObject( hSession, publicKeyTemplate, sizeof(
publicKeyTemplate ) / sizeof( CK_ATTRIBUTE ), &publicKeyHandle );
     assert(rv == CKR_OK);

     rv     =     pF->C_SignInit(hSession,     &signOnlyMechanism,
privateKeyHandle);
     assert(rv == CKR_OK);
     rv        =        pF->C_Sign(hSession,        pangramDigest,
sizeof(pangramDigest), NULL, &signatureLen);
     assert(rv == CKR_OK);
     assert(signatureLen == 128);
```

```
     rv       =       pF->C_Sign(hSession,       pangramDigest,
sizeof(pangramDigest), hashSignature, &signatureLen);
     assert(rv == CKR_OK);

     rv   =   pF->C_SignInit(hSession,   &signWithHashMechanism,
privateKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Sign(hSession, pangram, sizeof(pangram) - 1, NULL,
&signatureLen);
     assert(rv == CKR_OK);
     assert(signatureLen == 128);
     rv  =  pF->C_Sign(hSession,  pangram,  sizeof(pangram)  -  1,
dataSignature, &signatureLen);
     assert(rv == CKR_OK);

     rv    =    pF->C_VerifyInit(hSession,    &signOnlyMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv       =       pF->C_Verify(hSession,       pangramDigest,
sizeof(pangramDigest), hashSignature, signatureLen);
     assert(rv == CKR_OK);
     rv    =    pF->C_VerifyInit(hSession,    &signOnlyMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv       =       pF->C_Verify(hSession,       pangramDigest,
sizeof(pangramDigest), dataSignature, signatureLen);
     assert(rv == CKR_OK);
     rv    =    pF->C_VerifyInit(hSession,    &signOnlyMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv       =       pF->C_Verify(hSession,       pangramDigest,
sizeof(pangramDigest), ETALON, signatureLen);
     assert(rv == CKR_OK);

     rv   =   pF->C_VerifyInit(hSession,   &signWithHashMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv  =  pF->C_Verify(hSession,  pangram,  sizeof(pangram)  -  1,
hashSignature, signatureLen);
     assert(rv == CKR_OK);
     rv   =   pF->C_VerifyInit(hSession,   &signWithHashMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_Verify(hSession, pangram, sizeof(pangram) - 1,
dataSignature, signatureLen);
     assert(rv == CKR_OK);
     rv   =   pF->C_VerifyInit(hSession,   &signWithHashMechanism,
publicKeyHandle);
     assert(rv == CKR_OK);
     rv  =  pF->C_Verify(hSession,  pangram,  sizeof(pangram)  -  1,
ETALON, signatureLen);
     assert(rv == CKR_OK);
```

```
	rv = pF->C_DestroyObject(hSession, publicKeyHandle);
	assert(rv == CKR_OK);
	rv = pF->C_DestroyObject(hSession, privateKeyHandle);
	assert(rv == CKR_OK);

	return CKR_OK;
}
```

Приложение 3.13 (справочное). Пример использования механизма CKM_GOSTR3410_2012_DERIVE с ключом длины 256 бит для выработки ключа обмена.

```
CK_RV  sample_dh_2012(CK_FUNCTION_LIST_PTR  pF,  CK_SESSION_HANDLE
hSession)
{
	CK_RV rv = CKR_OK;
	CK_OBJECT_HANDLE aliceKeyHandle = CK_INVALID_HANDLE;
	CK_OBJECT_HANDLE secretKeyHandle = CK_INVALID_HANDLE;

	CK_BYTE ukm[] = {1, 2, 3, 4, 5, 6, 7, 8};
	CK_BYTE aliceKeyValue[] = {
		0xD9, 0x2D, 0x43, 0x1D, 0x20, 0x37, 0x5C, 0xD2,
		0xA5, 0x37, 0xCD, 0x64, 0x8E, 0x14, 0xB6, 0x0B,
		0x4C, 0x21, 0xA1, 0x5A, 0x57, 0x98, 0x61, 0xB7,
		0xBE, 0x41, 0x9B, 0x16, 0xED, 0x86, 0x18, 0x74,
	};
	CK_BYTE bobKeyValue[] = {
		0x4F, 0xC5, 0xF5, 0x7A, 0xB0, 0x9A, 0xA6, 0xF0,
		0xF7, 0x43, 0x3E, 0xDE, 0xFB, 0xB4, 0xBC, 0xBE,
		0x43, 0x68, 0xD6, 0x4F, 0xCF, 0x5E, 0xC6, 0x94,
		0x52, 0x98, 0x2C, 0xFA, 0xEF, 0x61, 0xFD, 0xC6,
		0xAE, 0x37, 0x76, 0x4B, 0xC9, 0xF9, 0x10, 0x90,
		0x59, 0x95, 0xE9, 0x23, 0x89, 0x53, 0x7F, 0xF3,
		0xB6, 0x32, 0x93, 0x8A, 0x4A, 0x6B, 0x8E, 0x5D,
		0x1B, 0xEE, 0x20, 0xDE, 0xE3, 0x71, 0xE2, 0x58,
	};
	CK_BYTE ETALON[] = {
		0x13, 0xC0, 0xBC, 0xD5, 0x1E, 0x44, 0x9B, 0x2C,
		0x20, 0x5D, 0xF0, 0x5D, 0xFD, 0xCA, 0xCE, 0xE8,
		0x54, 0xB6, 0xC4, 0xE9, 0xC5, 0x57, 0xF1, 0x3A,
		0xCF, 0xF6, 0x03, 0x93, 0x59, 0x92, 0xF9, 0xC2,
	};

	CK_BYTE gost3410_defOid[] = { 0x06, 0x07, 0x2a, 0x85, 0x03,
0x02, 0x02, 0x23, 0x01 };
	CK_BBOOL bTrue = CK_TRUE;
	CK_BBOOL bFalse = CK_FALSE;
	CK_KEY_TYPE secretType = CKK_GOST28147;
	CK_KEY_TYPE privateType = CKK_GOSTR3410;
	CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
	CK_OBJECT_CLASS secretClass = CKO_SECRET_KEY;
	CK_ATTRIBUTE aliceKeyTemplate[] = {
```

```
          {  CKA_CLASS,                              &privateClass,
sizeof(privateClass) },
          {   CKA_KEY_TYPE,                           &privateType,
sizeof(privateType) },
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bTrue,          sizeof(bTrue)
},
          {  CKA_VALUE,                              aliceKeyValue,
sizeof(aliceKeyValue) },
          { CKA_DERIVE,            &bTrue,          sizeof(bTrue)
},
          {                                  CKA_GOSTR3410_PARAMS,
gost3410_defOid,sizeof(gost3410_defOid) },
     };
     CK_ULONG aliceKeyTemplateSize = sizeof( aliceKeyTemplate ) /
sizeof( CK_ATTRIBUTE );
     CK_ATTRIBUTE secretTemplate[] =
     {
          {  CKA_CLASS,                              &secretClass,
sizeof(secretClass)},
          { CKA_KEY_TYPE,       &secretType,   sizeof(secretType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bFalse,         sizeof(bFalse)
},
          { CKA_EXTRACTABLE,       &bTrue,          sizeof(bTrue)
},
     };
     CK_ULONG secretTemplateSize = sizeof( secretTemplate ) /
sizeof( CK_ATTRIBUTE );
     uint32_t deriveParams[21] = {0};
     CK_MECHANISM deriveMechanism = { CKM_GOSTR3410_2012_DERIVE,
&deriveParams, sizeof( deriveParams ) };
     CK_BYTE secretKeyValue[32];
     CK_ATTRIBUTE keyAttribute = {CKA_VALUE, secretKeyValue,
sizeof(secretKeyValue)};

     rv = pF->C_CreateObject( hSession, aliceKeyTemplate,
aliceKeyTemplateSize, &aliceKeyHandle );
     assert(rv == CKR_OK);

     deriveParams[0] = CKD_NULL;
     deriveParams[1] = 64;
     memcpy( deriveParams + 2, bobKeyValue, sizeof(bobKeyValue) );
     deriveParams[18] = 8;
     memcpy( deriveParams + 19, ukm, sizeof( ukm ) );
     rv   =   pF->C_DeriveKey(   hSession,   &deriveMechanism,
aliceKeyHandle,       secretTemplate,       secretTemplateSize,
&secretKeyHandle );
     assert(rv == CKR_OK);
```

```
    rv = pF->C_GetAttributeValue(hSession, secretKeyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, aliceKeyHandle);
    assert(rv == CKR_OK);
    rv = pF->C_DestroyObject(hSession, secretKeyHandle);
    assert(rv == CKR_OK);

    return memcmp(secretKeyValue, ETALON, sizeof(ETALON)) ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.14 (справочное). Пример использования механизма CKM_GOSTR3410_2012_DERIVE с ключом длины 512 бит для выработки ключа обмена.

```
CK_RV         sample_dh_2012_512(CK_FUNCTION_LIST_PTR         pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv = CKR_OK;
    CK_OBJECT_HANDLE aliceKeyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE secretKeyHandle = CK_INVALID_HANDLE;

    CK_BYTE ukm[] = {1, 2, 3, 4, 5, 6, 7, 8};
    CK_BYTE aliceKeyValue[] = {
        0xC2, 0x48, 0x02, 0x82, 0x70, 0xE0, 0xFF, 0x17,
        0xD4, 0xDD, 0x9D, 0xA7, 0x19, 0xE2, 0xBD, 0xB6,
        0xDF, 0x60, 0x17, 0x2B, 0xCB, 0xC1, 0x70, 0x9A,
        0xBC, 0x4B, 0xAA, 0x80, 0xD2, 0xB6, 0x56, 0x9B,
        0x69, 0xDC, 0xED, 0x7A, 0x02, 0x66, 0xAC, 0xE0,
        0xA2, 0x64, 0x2C, 0xB4, 0x3A, 0x35, 0x87, 0x8F,
        0x82, 0x5F, 0x30, 0x2F, 0x14, 0x63, 0xDE, 0xC0,
        0xB7, 0x41, 0x33, 0xAF, 0x55, 0x81, 0x65, 0x40,
    };
    CK_BYTE bobKeyValue[] = {
        0x73, 0x50, 0x68, 0x39, 0x15, 0x51, 0x22, 0x45,
        0x0C, 0x15, 0x30, 0x88, 0xCD, 0xA0, 0x1A, 0xBC,
        0xBE, 0xA0, 0x4D, 0x9B, 0x3F, 0xCC, 0xB1, 0xB6,
        0x95, 0xF9, 0x49, 0x63, 0x0D, 0x02, 0xEF, 0xFE,
        0x0D, 0xA2, 0xC2, 0xCB, 0x84, 0x88, 0x43, 0x7B,
        0x05, 0x03, 0xB3, 0x31, 0x43, 0x0E, 0xD1, 0x6D,
        0xFF, 0xB9, 0x11, 0x1D, 0x44, 0xF1, 0x35, 0x23,
        0xF1, 0x38, 0x5B, 0x79, 0x03, 0x17, 0xB8, 0xEE,
        0x9B, 0x2E, 0xC2, 0x56, 0x6F, 0x78, 0xD6, 0xB1,
        0xF7, 0xDB, 0xD7, 0xC8, 0xBE, 0x89, 0x33, 0x8D,
        0x72, 0xD4, 0x1E, 0x4A, 0x60, 0x11, 0x0E, 0x16,
        0x4A, 0x01, 0x3C, 0x52, 0xBF, 0xF5, 0x8C, 0x9B,
        0x83, 0xB8, 0xDB, 0x64, 0xFB, 0xA1, 0xE7, 0x03,
        0x64, 0xD5, 0x26, 0xF6, 0x79, 0x3C, 0x4E, 0x35,
        0x5F, 0x58, 0x87, 0x69, 0x59, 0x28, 0xFA, 0x1F,
        0xCC, 0x20, 0x0F, 0x42, 0x46, 0x55, 0xF3, 0x95,
```

```
     };
     CK_BYTE ETALON[] = {
          0xef, 0xc6, 0x4a, 0x95, 0x35, 0x7b, 0xb7, 0x21,
          0x57, 0x6f, 0x25, 0xbd, 0x2a, 0xb9, 0x22, 0xf1,
          0x16, 0x69, 0xf3, 0xb1, 0xa2, 0x32, 0xd4, 0x7b,
          0xae, 0xb9, 0x2a, 0xaf, 0xa6, 0x10, 0x25, 0x64,
     };
     CK_BYTE gost3410_defOid[] = { 0x06, 0x09, 0x2a, 0x85, 0x03,
0x07, 0x01, 0x02, 0x01, 0x02, 0x01 }; //1.2.643.7.1.2.1.2.1
     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE secretType = CKK_GOST28147;
     CK_KEY_TYPE privateType = CKK_GOSTR3410_512;
     CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
     CK_OBJECT_CLASS secretClass = CKO_SECRET_KEY;
     CK_ATTRIBUTE aliceKeyTemplate[] = {
          {  CKA_CLASS,                              &privateClass,
sizeof(privateClass) },
          {   CKA_KEY_TYPE,                           &privateType,
sizeof(privateType) },
          { CKA_TOKEN,            &bFalse,        sizeof(bFalse)
},
          { CKA_PRIVATE,           &bTrue,          sizeof(bTrue)
},
          {  CKA_VALUE,                              aliceKeyValue,
sizeof(aliceKeyValue) },
          { CKA_DERIVE,            &bTrue,          sizeof(bTrue)
},
          {                                  CKA_GOSTR3410_PARAMS,
gost3410_defOid,sizeof(gost3410_defOid) },
     };
     CK_ULONG aliceKeyTemplateSize = sizeof( aliceKeyTemplate ) /
sizeof( CK_ATTRIBUTE );
     CK_ATTRIBUTE secretTemplate[] =
     {
          {  CKA_CLASS,                              &secretClass,
sizeof(secretClass)},
          { CKA_KEY_TYPE,       &secretType,   sizeof(secretType)
},
          { CKA_TOKEN,            &bFalse,        sizeof(bFalse)
},
          { CKA_PRIVATE,          &bFalse,        sizeof(bFalse)
},
          { CKA_EXTRACTABLE,       &bTrue,          sizeof(bTrue)
},
     };
     CK_ULONG secretTemplateSize = sizeof( secretTemplate ) /
sizeof( CK_ATTRIBUTE );
     uint32_t deriveParams[37] = {0};
     CK_MECHANISM deriveMechanism = { CKM_GOSTR3410_2012_DERIVE,
&deriveParams, sizeof( deriveParams ) };
     CK_BYTE secretKeyValue[32];
```

```
    CK_ATTRIBUTE keyAttribute = {CKA_VALUE, secretKeyValue,
sizeof(secretKeyValue)};

    rv = pF->C_CreateObject( hSession, aliceKeyTemplate,
aliceKeyTemplateSize, &aliceKeyHandle );
    assert(rv == CKR_OK);

    deriveParams[0] = CKD_NULL;
    deriveParams[1] = 128;
    memcpy( deriveParams + 2, bobKeyValue, sizeof(bobKeyValue) );
    deriveParams[34] = 8;
    memcpy( deriveParams + 35, ukm, sizeof( ukm ) );
    rv = pF->C_DeriveKey( hSession, &deriveMechanism,
aliceKeyHandle, secretTemplate, secretTemplateSize,
&secretKeyHandle );
    assert(rv == CKR_OK);

    rv = pF->C_GetAttributeValue(hSession, secretKeyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, aliceKeyHandle);
    assert(rv == CKR_OK);
    rv = pF->C_DestroyObject(hSession, secretKeyHandle);
    assert(rv == CKR_OK);

    return memcmp(secretKeyValue, ETALON, sizeof(ETALON)) ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.15 (справочное). Пример использования механизма CKM_VKO_GOSTR3410_2012_512.

```
CK_RV       sample_dh_2012_512_512(CK_FUNCTION_LIST_PTR       pF,
CK_SESSION_HANDLE hSession)
{
    CK_RV rv;
    CK_BYTE ukm[] = {
            0x1d, 0x80, 0x60, 0x3c, 0x85, 0x44, 0xc7, 0x27,
    };
    CK_BYTE aliceKeyValue[] = {
            0xc9, 0x90, 0xec, 0xd9, 0x72, 0xfc, 0xe8, 0x4e,
            0xc4, 0xdb, 0x02, 0x27, 0x78, 0xf5, 0x0f, 0xca,
            0xc7, 0x26, 0xf4, 0x67, 0x08, 0x38, 0x4b, 0x8d,
            0x45, 0x83, 0x04, 0x96, 0x2d, 0x71, 0x47, 0xf8,
            0xc2, 0xdb, 0x41, 0xce, 0xf2, 0x2c, 0x90, 0xb1,
            0x02, 0xf2, 0x96, 0x84, 0x04, 0xf9, 0xb9, 0xbe,
            0x6d, 0x47, 0xc7, 0x96, 0x92, 0xd8, 0x18, 0x26,
            0xb3, 0x2b, 0x8d, 0xac, 0xa4, 0x3c, 0xb6, 0x67,
    };
    CK_BYTE bobKeyValue[] = {
            0x19, 0x2f, 0xe1, 0x83, 0xb9, 0x71, 0x3a, 0x07,
            0x72, 0x53, 0xc7, 0x2c, 0x87, 0x35, 0xde, 0x2e,
```

```
            0xa4, 0x2a, 0x3d, 0xbc, 0x66, 0xea, 0x31, 0x78,
            0x38, 0xb6, 0x5f, 0xa3, 0x25, 0x23, 0xcd, 0x5e,
            0xfc, 0xa9, 0x74, 0xed, 0xa7, 0xc8, 0x63, 0xf4,
            0x95, 0x4d, 0x11, 0x47, 0xf1, 0xf2, 0xb2, 0x5c,
            0x39, 0x5f, 0xce, 0x1c, 0x12, 0x91, 0x75, 0xe8,
            0x76, 0xd1, 0x32, 0xe9, 0x4e, 0xd5, 0xa6, 0x51,
            0x04, 0x88, 0x3b, 0x41, 0x4c, 0x9b, 0x59, 0x2e,
            0xc4, 0xdc, 0x84, 0x82, 0x6f, 0x07, 0xd0, 0xb6,
            0xd9, 0x00, 0x6d, 0xda, 0x17, 0x6c, 0xe4, 0x8c,
            0x39, 0x1e, 0x3f, 0x97, 0xd1, 0x02, 0xe0, 0x3b,
            0xb5, 0x98, 0xbf, 0x13, 0x2a, 0x22, 0x8a, 0x45,
            0xf7, 0x20, 0x1a, 0xba, 0x08, 0xfc, 0x52, 0x4a,
            0x2d, 0x77, 0xe4, 0x3a, 0x36, 0x2a, 0xb0, 0x22,
            0xad, 0x40, 0x28, 0xf7, 0x5b, 0xde, 0x3b, 0x79,
      };
      CK_BYTE ETALON[] = {
            0x79, 0xf0, 0x02, 0xa9, 0x69, 0x40, 0xce, 0x7b,
            0xde, 0x32, 0x59, 0xa5, 0x2e, 0x01, 0x52, 0x97,
            0xad, 0xaa, 0xd8, 0x45, 0x97, 0xa0, 0xd2, 0x05,
            0xb5, 0x0e, 0x3e, 0x17, 0x19, 0xf9, 0x7b, 0xfa,
            0x7e, 0xe1, 0xd2, 0x66, 0x1f, 0xa9, 0x97, 0x9a,
            0x5a, 0xa2, 0x35, 0xb5, 0x58, 0xa7, 0xe6, 0xd9,
            0xf8, 0x8f, 0x98, 0x2d, 0xd6, 0x3f, 0xc3, 0x5a,
            0x8e, 0xc0, 0xdd, 0x5e, 0x24, 0x2d, 0x3b, 0xdf,
      };

      // 1.2.643.7.1.2.1.2.1  id-tc26-gost-3410-12-512-paramSetA
      CK_BYTE gost3410_defOid[] = {
            0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01,
0x02, 0x01
      };
      CK_BBOOL bTrue = CK_TRUE;
      CK_BBOOL bFalse = CK_FALSE;
      CK_KEY_TYPE secretType = CKK_MAGMA_TWIN_KEY;
      CK_KEY_TYPE privateType = CKK_GOSTR3410_512;
      CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
      CK_OBJECT_CLASS secretClass = CKO_SECRET_KEY;
      CK_ATTRIBUTE aliceKeyTemplate[] = {
           {  CKA_CLASS,                              &privateClass,
sizeof(privateClass) },
           {   CKA_KEY_TYPE,                           &privateType,
sizeof(privateType) },
           { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
           { CKA_PRIVATE,            &bTrue,          sizeof(bTrue)
},
           {  CKA_VALUE,                              aliceKeyValue,
sizeof(aliceKeyValue) },
           { CKA_DERIVE,             &bTrue,          sizeof(bTrue)
},
           {                                  CKA_GOSTR3410_PARAMS,
gost3410_defOid,sizeof(gost3410_defOid) },
      };
```

```
     CK_ULONG aliceKeyTemplateSize = sizeof( aliceKeyTemplate ) /
sizeof( CK_ATTRIBUTE );
     CK_ATTRIBUTE secretTemplate[] =
     {
          {  CKA_CLASS,                              &secretClass,
sizeof(secretClass)},
          { CKA_KEY_TYPE,       &secretType,   sizeof(secretType)
},
          { CKA_TOKEN,             &bFalse,        sizeof(bFalse)
},
          { CKA_PRIVATE,           &bFalse,        sizeof(bFalse)
},
          { CKA_EXTRACTABLE,       &bTrue,          sizeof(bTrue)
},
          { CKA_SENSITIVE,         &bFalse,        sizeof(bFalse)
},
     };
     CK_ULONG  secretTemplateSize  =  sizeof(  secretTemplate  )  /
sizeof( CK_ATTRIBUTE );

     CK_ECDH1_DERIVE_PARAMS mechanismParam = {
          CKD_NULL,
          sizeof(ukm), ukm,
          sizeof(bobKeyValue), bobKeyValue };
     CK_MECHANISM deriveMechanism = { CKM_VKO_GOSTR3410_2012_512,
&mechanismParam, sizeof( mechanismParam ) };

     CK_BYTE secretKeyValue[sizeof(ETALON)];
     CK_ATTRIBUTE  keyAttribute  =  {CKA_VALUE,  secretKeyValue,
sizeof(secretKeyValue)};

     CK_OBJECT_HANDLE aliceKeyHandle = CK_INVALID_HANDLE;
     CK_OBJECT_HANDLE secretKeyHandle = CK_INVALID_HANDLE;

     rv   =   pF->C_CreateObject(   hSession,   aliceKeyTemplate,
aliceKeyTemplateSize, &aliceKeyHandle );
     assert(rv == CKR_OK);

     rv    =    pF->C_DeriveKey(    hSession,    &deriveMechanism,
aliceKeyHandle,        secretTemplate,        secretTemplateSize,
&secretKeyHandle );
     assert(rv == CKR_OK);

     rv   =   pF->C_GetAttributeValue(hSession,   secretKeyHandle,
&keyAttribute, 1);
     assert(rv == CKR_OK);

     rv = pF->C_DestroyObject(hSession, aliceKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_DestroyObject(hSession, secretKeyHandle);
     assert(rv == CKR_OK);
```

```
     return  memcmp(secretKeyValue,  ETALON,  sizeof(ETALON))  ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.16 (справочное). Пример использования механизма CKM_GOST_KEG с ключом длины 256 бит.

```
CK_RV  sample_keg_256(CK_FUNCTION_LIST_PTR  pF,  CK_SESSION_HANDLE
hSession)
{
     CK_RV rv;

     CK_BYTE ukm[] = {
            0xC3, 0xEF, 0x04, 0x28, 0xD4, 0xB7, 0xA1, 0xF4,
            0xC5, 0x02, 0x5F, 0x2E, 0x65, 0xDD, 0x2B, 0x2E,
            0xA5, 0x83, 0xAE, 0xEF, 0xDB, 0x67, 0xC7, 0xF4,
            0x21, 0x4A, 0x6A, 0x29, 0x8E, 0x99, 0xE3, 0x25,
     };
     CK_BYTE aliceKeyValue[] = {
            0xE4, 0xE0, 0x96, 0x73, 0x1B, 0x27, 0xAB, 0x36,
            0x13, 0xD9, 0x56, 0xBF, 0x6A, 0xE1, 0x83, 0x55,
            0xB5, 0x04, 0x3A, 0x55, 0x10, 0xA3, 0x43, 0xA1,
            0x6C, 0x65, 0xC7, 0x94, 0x1E, 0xFD, 0x80, 0x08,
     };
     CK_BYTE bobKeyValue[] = {
            0x8D, 0x49, 0x0F, 0x4C, 0xB0, 0x30, 0xE2, 0x39,
            0x74, 0xE2, 0x18, 0xC2, 0x78, 0x73, 0x12, 0xBE,
            0xD9, 0xF3, 0x61, 0x37, 0x7C, 0xCC, 0xF5, 0x2A,
            0x7E, 0x73, 0x85, 0x6C, 0x2A, 0x19, 0xD9, 0x86,
            0x00, 0xCE, 0xC1, 0xB8, 0x36, 0xC6, 0x40, 0x5B,
            0x24, 0x1B, 0xA7, 0xCD, 0x8C, 0x08, 0x5E, 0x2D,
            0xEC, 0x6C, 0x4E, 0x0A, 0x61, 0xD9, 0x72, 0xF1,
            0xD9, 0x8F, 0xE4, 0xB8, 0x76, 0x0E, 0x19, 0x71,
     };
     CK_BYTE ETALON[] = {
            0x30, 0x95, 0x48, 0x96, 0x6C, 0x4A, 0x84, 0xA8,
            0xA7, 0x7C, 0x1B, 0xFF, 0x99, 0x63, 0xCD, 0xE7,
            0xAF, 0x76, 0xDB, 0x9C, 0xF1, 0xD1, 0x40, 0x01,
            0x97, 0x13, 0x6F, 0x23, 0x99, 0x89, 0x20, 0x02,
            0xC1, 0x17, 0x69, 0x2B, 0x8C, 0xF5, 0xE7, 0x89,
            0x75, 0xD4, 0x21, 0xC4, 0x7E, 0x6D, 0x36, 0xBB,
            0xD0, 0x03, 0x76, 0x8F, 0x93, 0x4E, 0xBB, 0x6C,
            0x98, 0xA2, 0xCD, 0xC9, 0x57, 0x6B, 0x5C, 0x67,
     };

      // "1.2.643.7.1.2.1.1.1"
     CK_BYTE gost3410_defOid[] = {
            0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01,
0x01, 0x01
     };
     // "1.2.643.7.1.1.2.2"
     CK_BYTE stribog256Oid[] = {
```

```
            0x06, 0x08, 0x2a, (byte) 0x85, 0x03, 0x07, 0x01, 0x01,
0x02, 0x02,
     };
     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE secretType = CKK_MAGMA_TWIN_KEY;
     CK_KEY_TYPE privateType = CKK_GOSTR3410;
     CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
     CK_OBJECT_CLASS secretClass = CKO_SECRET_KEY;
     CK_ATTRIBUTE aliceKeyTemplate[] = {
          {   CKA_CLASS,                              &privateClass,
sizeof(privateClass) },
          {   CKA_KEY_TYPE,                            &privateType,
sizeof(privateType) },
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bTrue,           sizeof(bTrue)
},
          {   CKA_VALUE,                              aliceKeyValue,
sizeof(aliceKeyValue) },
          { CKA_DERIVE,            &bTrue,           sizeof(bTrue)
},
          {                                  CKA_GOSTR3410_PARAMS,
gost3410_defOid,sizeof(gost3410_defOid) },
          {          CKA_GOSTR3411_PARAMS,          stribog256Oid,
sizeof(stribog256Oid) },
     };
     CK_ULONG aliceKeyTemplateSize = sizeof( aliceKeyTemplate ) /
sizeof( CK_ATTRIBUTE );
     CK_ATTRIBUTE secretTemplate[] =
     {
          {   CKA_CLASS,                               &secretClass,
sizeof(secretClass)},
          { CKA_KEY_TYPE,       &secretType,   sizeof(secretType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bFalse,         sizeof(bFalse)
},
          { CKA_EXTRACTABLE,       &bTrue,           sizeof(bTrue)
},
          { CKA_SENSITIVE,         &bFalse,         sizeof(bFalse)
},
     };
     CK_ULONG secretTemplateSize = sizeof( secretTemplate ) /
sizeof( CK_ATTRIBUTE );

     CK_ECDH1_DERIVE_PARAMS mechanismParam = {
          CKD_NULL,
          sizeof(ukm), ukm,
          sizeof(bobKeyValue), bobKeyValue };
     CK_MECHANISM    deriveMechanism    =    {    CKM_GOST_KEG,
&mechanismParam, sizeof( mechanismParam ) };
```

```
    CK_BYTE secretKeyValue[sizeof(ETALON)];
    CK_ATTRIBUTE keyAttribute = {CKA_VALUE, secretKeyValue,
sizeof(secretKeyValue)};

    CK_OBJECT_HANDLE aliceKeyHandle = CK_INVALID_HANDLE;
    CK_OBJECT_HANDLE secretKeyHandle = CK_INVALID_HANDLE;

    rv = pF->C_CreateObject( hSession, aliceKeyTemplate,
aliceKeyTemplateSize, &aliceKeyHandle );
    assert(rv == CKR_OK);

    rv = pF->C_DeriveKey( hSession, &deriveMechanism,
aliceKeyHandle, secretTemplate, secretTemplateSize,
&secretKeyHandle );
    assert(rv == CKR_OK);

    rv = pF->C_GetAttributeValue(hSession, secretKeyHandle,
&keyAttribute, 1);
    assert(rv == CKR_OK);

    rv = pF->C_DestroyObject(hSession, aliceKeyHandle);
    assert(rv == CKR_OK);
    rv = pF->C_DestroyObject(hSession, secretKeyHandle);
    assert(rv == CKR_OK);

    return memcmp(secretKeyValue, ETALON, sizeof(ETALON)) ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.17 (справочное). Пример использования механизма CKM_GOST_KEG с ключом длины 512 бит.

```
CK_RV sample_keg_512(CK_FUNCTION_LIST_PTR pF, CK_SESSION_HANDLE
hSession)
{
    CK_RV rv;

    CK_BYTE ukm[] = {
          0xC3, 0xEF, 0x04, 0x28, 0xD4, 0xB7, 0xA1, 0xF4,
          0xC5, 0x02, 0x5F, 0x2E, 0x65, 0xDD, 0x2B, 0x2E,
          0xA5, 0x83, 0xAE, 0xEF, 0xDB, 0x67, 0xC7, 0xF4,
          0x21, 0x4A, 0x6A, 0x29, 0x8E, 0x99, 0xE3, 0x25,
    };
    CK_BYTE aliceKeyValue[] = {
          0x01, 0x2E, 0x19, 0x45, 0xD4, 0xC6, 0x1A, 0x8E,
          0x52, 0xCF, 0x37, 0xE1, 0xBF, 0xFB, 0x7D, 0xD2,
          0x8D, 0xC0, 0x5F, 0x0D, 0x64, 0xAE, 0xCC, 0xA8,
          0x89, 0x89, 0x2D, 0x5A, 0x36, 0xED, 0x05, 0x30,
          0x4B, 0x64, 0x01, 0x26, 0x40, 0x8B, 0x6F, 0x80,
          0x79, 0x2D, 0xC7, 0x3C, 0xFD, 0xAB, 0xC7, 0xFB,
          0xAD, 0x34, 0x55, 0xA2, 0xF9, 0x59, 0x6C, 0xF6,
          0xA0, 0x79, 0x74, 0x06, 0x70, 0x7A, 0xFD, 0x12,
```

```
    };
    CK_BYTE bobKeyValue[] = {
           0xC6, 0x5B, 0xD7, 0x05, 0xB6, 0x86, 0x01, 0x98,
           0xBA, 0xD4, 0xA7, 0x0E, 0xB9, 0x37, 0xB6, 0xB4,
           0x80, 0x84, 0xE2, 0x60, 0xAD, 0xF7, 0xB1, 0x07,
           0x4A, 0x89, 0x18, 0x28, 0x62, 0xC5, 0xBF, 0xFE,
           0x64, 0x86, 0x28, 0x35, 0x41, 0x33, 0x0B, 0x15,
           0x0F, 0xE4, 0x8A, 0x73, 0x7C, 0xB3, 0xE5, 0xBB,
           0x04, 0x3E, 0x4A, 0x11, 0x34, 0x03, 0x5A, 0x6D,
           0x47, 0x9B, 0x18, 0x93, 0x51, 0xBE, 0x41, 0xC9,
           0xBE, 0x9A, 0x7E, 0x2A, 0xFC, 0x24, 0x62, 0x76,
           0xFE, 0x4E, 0x23, 0x56, 0x84, 0x52, 0x93, 0xB0,
           0x31, 0x78, 0xE2, 0xEC, 0x00, 0x3C, 0xA8, 0xA8,
           0x14, 0x32, 0x4F, 0x16, 0x35, 0x0B, 0xC0, 0xAB,
           0x53, 0x41, 0x87, 0xDE, 0x86, 0xC7, 0x6B, 0xE2,
           0x9A, 0x94, 0x0A, 0x8D, 0xB2, 0xAD, 0x71, 0x64,
           0x6A, 0xA0, 0xC9, 0x52, 0xFD, 0xF4, 0x11, 0x20,
           0x65, 0x48, 0x81, 0x3E, 0xB9, 0xF7, 0x54, 0xA1,
    };
    CK_BYTE ETALON[] = {
           0x7D, 0xAC, 0x56, 0xE4, 0x8A, 0x4D, 0xC1, 0x70,
           0xFA, 0xA8, 0xFC, 0xBA, 0xE2, 0x0D, 0xB8, 0x45,
           0x45, 0x0C, 0xCC, 0xC4, 0xC6, 0x32, 0x8B, 0xDC,
           0x8D, 0x01, 0x15, 0x7C, 0xEF, 0xA2, 0xA5, 0xF1,
           0x1F, 0x1C, 0xBA, 0xD8, 0x86, 0x61, 0x66, 0xF0,
           0x1F, 0xFA, 0xAB, 0x01, 0x52, 0xE2, 0x4B, 0xF4,
           0x60, 0x9D, 0x5F, 0x46, 0xA5, 0xC8, 0x99, 0xC7,
           0x87, 0x90, 0x0D, 0x08, 0xB9, 0xFC, 0xAD, 0x24,
    };

     // "1.2.643.7.1.2.1.2.3";
    CK_BYTE gost3410_defOid[] = {
           0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01,
0x02, 0x03
    };
    CK_BBOOL bTrue = CK_TRUE;
    CK_BBOOL bFalse = CK_FALSE;
    CK_KEY_TYPE secretType = CKK_KUZNECHIK_TWIN_KEY;
    CK_KEY_TYPE privateType = CKK_GOSTR3410_512;
    CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
    CK_OBJECT_CLASS secretClass = CKO_SECRET_KEY;
    CK_ATTRIBUTE aliceKeyTemplate[] = {
         {  CKA_CLASS,                              &privateClass,
sizeof(privateClass) },
         {   CKA_KEY_TYPE,                           &privateType,
sizeof(privateType) },
         { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
         { CKA_PRIVATE,           &bTrue,          sizeof(bTrue)
},
         {  CKA_VALUE,                              aliceKeyValue,
sizeof(aliceKeyValue) },
```

```
          { CKA_DERIVE,            &bTrue,          sizeof(bTrue)
},
          {                                  CKA_GOSTR3410_PARAMS,
gost3410_defOid,sizeof(gost3410_defOid) },
     };
     CK_ULONG aliceKeyTemplateSize = sizeof( aliceKeyTemplate ) /
sizeof( CK_ATTRIBUTE );
     CK_ATTRIBUTE secretTemplate[] =
     {
          {  CKA_CLASS,                              &secretClass,
sizeof(secretClass)},
          { CKA_KEY_TYPE,       &secretType,   sizeof(secretType)
},
          { CKA_TOKEN,            &bFalse,        sizeof(bFalse)
},
          { CKA_PRIVATE,          &bFalse,        sizeof(bFalse)
},
          { CKA_EXTRACTABLE,       &bTrue,         sizeof(bTrue)
},
          { CKA_SENSITIVE,        &bFalse,        sizeof(bFalse)
},
     };
     CK_ULONG secretTemplateSize = sizeof( secretTemplate ) /
sizeof( CK_ATTRIBUTE );

     CK_ECDH1_DERIVE_PARAMS mechanismParam = {
          CKD_NULL,
          sizeof(ukm), ukm,
          sizeof(bobKeyValue), bobKeyValue };
     CK_MECHANISM    deriveMechanism    =    {    CKM_GOST_KEG,
&mechanismParam, sizeof( mechanismParam ) };

     CK_BYTE secretKeyValue[sizeof(ETALON)];
     CK_ATTRIBUTE  keyAttribute  =  {CKA_VALUE,  secretKeyValue,
sizeof(secretKeyValue)};

     CK_OBJECT_HANDLE aliceKeyHandle = CK_INVALID_HANDLE;
     CK_OBJECT_HANDLE secretKeyHandle = CK_INVALID_HANDLE;

     rv   =   pF->C_CreateObject(   hSession,   aliceKeyTemplate,
aliceKeyTemplateSize, &aliceKeyHandle );
     assert(rv == CKR_OK);

     rv    =    pF->C_DeriveKey(    hSession,    &deriveMechanism,
aliceKeyHandle,        secretTemplate,        secretTemplateSize,
&secretKeyHandle );
     assert(rv == CKR_OK);

     rv   =   pF->C_GetAttributeValue(hSession,   secretKeyHandle,
&keyAttribute, 1);
     assert(rv == CKR_OK);

     rv = pF->C_DestroyObject(hSession, aliceKeyHandle);
```

```
	assert(rv == CKR_OK);
	rv = pF->C_DestroyObject(hSession, secretKeyHandle);
	assert(rv == CKR_OK);

	return memcmp(secretKeyValue, ETALON, sizeof(ETALON)) ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.18 (справочное). Пример использования механизма CKM_ECDH1_DERIVE с ключом длины 256 бит.

```
CK_RV		sample_vko_ecdhe_256(CK_FUNCTION_LIST_PTR		pF,
CK_SESSION_HANDLE hSession)
{
	CK_RV rv;

	CK_BYTE aliceKeyValue[] = {
		0xE4, 0xE0, 0x96, 0x73, 0x1B, 0x27, 0xAB, 0x36,
		0x13, 0xD9, 0x56, 0xBF, 0x6A, 0xE1, 0x83, 0x55,
		0xB5, 0x04, 0x3A, 0x55, 0x10, 0xA3, 0x43, 0xA1,
		0x6C, 0x65, 0xC7, 0x94, 0x1E, 0xFD, 0x80, 0x08,
	};
	CK_BYTE bobKeyValue[] = {
		0x8D, 0x49, 0x0F, 0x4C, 0xB0, 0x30, 0xE2, 0x39,
		0x74, 0xE2, 0x18, 0xC2, 0x78, 0x73, 0x12, 0xBE,
		0xD9, 0xF3, 0x61, 0x37, 0x7C, 0xCC, 0xF5, 0x2A,
		0x7E, 0x73, 0x85, 0x6C, 0x2A, 0x19, 0xD9, 0x86,
		0x00, 0xCE, 0xC1, 0xB8, 0x36, 0xC6, 0x40, 0x5B,
		0x24, 0x1B, 0xA7, 0xCD, 0x8C, 0x08, 0x5E, 0x2D,
		0xEC, 0x6C, 0x4E, 0x0A, 0x61, 0xD9, 0x72, 0xF1,
		0xD9, 0x8F, 0xE4, 0xB8, 0x76, 0x0E, 0x19, 0x71,
	};
	CK_BYTE ETALON[] = {
		0x04, 0x0F, 0x27, 0x4C, 0x37, 0x69, 0x74, 0x3D,
		0x63, 0x74, 0x76, 0x4D, 0xFD, 0xBE, 0x42, 0x3D,
		0xC6, 0x73, 0x2B, 0x6D, 0xEF, 0xB6, 0xB5, 0x04,
		0xCF, 0xD3, 0x4F, 0x64, 0xB0, 0xE4, 0xB0, 0x58,
	};

	 // "1.2.643.7.1.2.1.1.1"
	CK_BYTE gost3410_defOid[] = {
		0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01,
0x01, 0x01
	};
	CK_BBOOL bTrue = CK_TRUE;
	CK_BBOOL bFalse = CK_FALSE;
	CK_KEY_TYPE secretType = CKK_MAGMA;
	CK_KEY_TYPE privateType = CKK_GOSTR3410;
	CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
	CK_OBJECT_CLASS secretClass = CKO_SECRET_KEY;
	CK_ATTRIBUTE aliceKeyTemplate[] = {
	    {  CKA_CLASS,                          &privateClass,
sizeof(privateClass) },
```

```
          {   CKA_KEY_TYPE,                          &privateType,
sizeof(privateType) },
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bTrue,          sizeof(bTrue)
},
          {  CKA_VALUE,                              aliceKeyValue,
sizeof(aliceKeyValue) },
          { CKA_DERIVE,            &bTrue,          sizeof(bTrue)
},
          {                                  CKA_GOSTR3410_PARAMS,
gost3410_defOid,sizeof(gost3410_defOid) },
     };
     CK_ULONG aliceKeyTemplateSize = sizeof( aliceKeyTemplate ) /
sizeof( CK_ATTRIBUTE );
     CK_ATTRIBUTE secretTemplate[] =
     {
          {  CKA_CLASS,                              &secretClass,
sizeof(secretClass)},
          { CKA_KEY_TYPE,       &secretType,   sizeof(secretType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bFalse,         sizeof(bFalse)
},
          { CKA_EXTRACTABLE,       &bTrue,          sizeof(bTrue)
},
          { CKA_SENSITIVE,         &bFalse,         sizeof(bFalse)
},
     };
     CK_ULONG secretTemplateSize = sizeof( secretTemplate ) /
sizeof( CK_ATTRIBUTE );

     CK_ECDH1_DERIVE_PARAMS mechanismParam = {
          CKD_NULL,
          0, NULL,
          sizeof(bobKeyValue), bobKeyValue };
     CK_MECHANISM   deriveMechanism   =   {   CKM_ECDH1_DERIVE,
&mechanismParam, sizeof( mechanismParam ) };

     CK_BYTE secretKeyValue[sizeof(ETALON)];
     CK_ATTRIBUTE  keyAttribute  =  {CKA_VALUE,  secretKeyValue,
sizeof(secretKeyValue)};

     CK_OBJECT_HANDLE aliceKeyHandle = CK_INVALID_HANDLE;
     CK_OBJECT_HANDLE secretKeyHandle = CK_INVALID_HANDLE;

     rv   =   pF->C_CreateObject(   hSession,   aliceKeyTemplate,
aliceKeyTemplateSize, &aliceKeyHandle );
     assert(rv == CKR_OK);
```

```
     rv    =    pF->C_DeriveKey(    hSession,    &deriveMechanism,
aliceKeyHandle,        secretTemplate,        secretTemplateSize,
&secretKeyHandle );
     assert(rv == CKR_OK);

     rv   =   pF->C_GetAttributeValue(hSession,   secretKeyHandle,
&keyAttribute, 1);
     assert(rv == CKR_OK);

     rv = pF->C_DestroyObject(hSession, aliceKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_DestroyObject(hSession, secretKeyHandle);
     assert(rv == CKR_OK);

     return  memcmp(secretKeyValue,  ETALON,  sizeof(ETALON))  ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```

Приложение 3.19 (справочное). Пример использования механизма CKM_ECDH1_DERIVE с ключом длины 512 бит.

```
CK_RV        sample_vko_ecdhe_512(CK_FUNCTION_LIST_PTR        pF,
CK_SESSION_HANDLE hSession)
{
     CK_RV rv;

     CK_BYTE aliceKeyValue[] = {
            0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,
            0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,
            0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,
            0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,
            0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,
            0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,
            0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,  0x04,
     };
     CK_BYTE bobKeyValue[] = {
            0x2F,  0x3C,  0x66,  0x3F,  0xE7,  0x47,  0x35,  0xA1,
            0xC4,  0x21,  0x16,  0x0D,  0xF0,  0xF4,  0x32,  0x66,
            0x18,  0x5F,  0xD3,  0x0B,  0x6E,  0x5D,  0x6E,  0x88,
            0xFC,  0x40,  0x61,  0xFA,  0xEA,  0xCA,  0xB3,  0x38,
            0xB1,  0x0A,  0x1B,  0xD2,  0x0C,  0xB0,  0xB4,  0xEE,
            0x75,  0x7E,  0x74,  0xA0,  0x02,  0x7D,  0x40,  0x9F,
            0xE9,  0x37,  0xF0,  0x16,  0x33,  0xA1,  0xE3,  0xF9,
            0xA5,  0x51,  0x8D,  0xEF,  0xD0,  0xF8,  0x9F,  0x9D,
            0x3D,  0x9F,  0x6C,  0xC6,  0x51,  0x41,  0x3D,  0xEC,
            0x2C,  0x74,  0x36,  0x6D,  0x83,  0xC4,  0x7E,  0xE1,
            0xDE,  0x4E,  0x42,  0x1F,  0x65,  0xCD,  0x11,  0x63,
            0xE9,  0x4E,  0xA0,  0xC2,  0xE1,  0x9E,  0xD4,  0x5D,
            0x35,  0x55,  0x8B,  0x93,  0x7D,  0x9B,  0xFD,  0xC5,
            0xEC,  0xC2,  0xB2,  0xA2,  0x1B,  0x4E,  0xC3,  0xD5,
            0x3B,  0x29,  0x57,  0x9A,  0x8F,  0xD5,  0xE0,  0x74,
            0x81,  0x10,  0x28,  0xFB,  0xCF,  0x17,  0x99,  0x4F,
```

```
     };
     CK_BYTE ETALON[] = {
            0x4D,  0xE6,  0x0D,  0x21,  0xEA,  0x8F,  0xB9,  0x22,
            0x0D,  0x14,  0x64,  0x23,  0xB4,  0x90,  0xDA,  0x40,
            0xCC,  0xEB,  0xC4,  0x3B,  0xC5,  0x89,  0xDB,  0x79,
            0xB8,  0x31,  0xA4,  0x7D,  0x6B,  0x06,  0x30,  0x07,
            0xDD,  0x03,  0x40,  0x5A,  0x1B,  0x79,  0x76,  0xB6,
            0x23,  0xDC,  0xAA,  0x69,  0xB0,  0x11,  0xAE,  0x10,
            0x6E,  0x7E,  0x41,  0x74,  0x38,  0x5F,  0x86,  0x26,
            0xE1,  0x21,  0xB5,  0x99,  0x43,  0x63,  0xC9,  0x9F,
     };

      // "1.2.643.7.1.2.1.2.3";
     CK_BYTE gost3410_defOid[] = {
            0x06, 0x09, 0x2a, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01,
0x02, 0x03
     };
     CK_BBOOL bTrue = CK_TRUE;
     CK_BBOOL bFalse = CK_FALSE;
     CK_KEY_TYPE secretType = CKK_GENERIC_SECRET;
     CK_KEY_TYPE privateType = CKK_GOSTR3410_512;
     CK_OBJECT_CLASS privateClass = CKO_PRIVATE_KEY;
     CK_OBJECT_CLASS secretClass = CKO_SECRET_KEY;
     CK_ULONG valueLength = 64;
     CK_ATTRIBUTE aliceKeyTemplate[] = {
          {  CKA_CLASS,                              &privateClass,
sizeof(privateClass) },
          {   CKA_KEY_TYPE,                           &privateType,
sizeof(privateType) },
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bTrue,          sizeof(bTrue)
},
          {  CKA_VALUE,                              aliceKeyValue,
sizeof(aliceKeyValue) },
          { CKA_DERIVE,            &bTrue,          sizeof(bTrue)
},
          {                                  CKA_GOSTR3410_PARAMS,
gost3410_defOid,sizeof(gost3410_defOid) },
     };
     CK_ULONG aliceKeyTemplateSize = sizeof( aliceKeyTemplate ) /
sizeof( CK_ATTRIBUTE );
     CK_ATTRIBUTE secretTemplate[] =
     {
          {  CKA_CLASS,                              &secretClass,
sizeof(secretClass)},
          { CKA_KEY_TYPE,       &secretType,   sizeof(secretType)
},
          { CKA_TOKEN,             &bFalse,         sizeof(bFalse)
},
          { CKA_PRIVATE,           &bFalse,         sizeof(bFalse)
},
```

```
          { CKA_EXTRACTABLE,       &bTrue,          sizeof(bTrue)
},
          { CKA_SENSITIVE,         &bFalse,         sizeof(bFalse)
},
          {   CKA_VALUE_LEN,                        &valueLength,
sizeof(valueLength) },
     };
     CK_ULONG secretTemplateSize = sizeof( secretTemplate ) /
sizeof( CK_ATTRIBUTE );

     CK_ECDH1_DERIVE_PARAMS mechanismParam = {
          CKD_NULL,
          0, NULL,
          sizeof(bobKeyValue), bobKeyValue };
     CK_MECHANISM   deriveMechanism   =   {   CKM_ECDH1_DERIVE,
&mechanismParam, sizeof( mechanismParam ) };

     CK_BYTE secretKeyValue[sizeof(ETALON)];
     CK_ATTRIBUTE  keyAttribute  =  {CKA_VALUE,  secretKeyValue,
sizeof(secretKeyValue)};

     CK_OBJECT_HANDLE aliceKeyHandle = CK_INVALID_HANDLE;
     CK_OBJECT_HANDLE secretKeyHandle = CK_INVALID_HANDLE;

     rv   =   pF->C_CreateObject(   hSession,   aliceKeyTemplate,
aliceKeyTemplateSize, &aliceKeyHandle );
     assert(rv == CKR_OK);

     rv    =    pF->C_DeriveKey(    hSession,    &deriveMechanism,
aliceKeyHandle,        secretTemplate,        secretTemplateSize,
&secretKeyHandle );
     assert(rv == CKR_OK);

     rv   =   pF->C_GetAttributeValue(hSession,   secretKeyHandle,
&keyAttribute, 1);
     assert(rv == CKR_OK);

     rv = pF->C_DestroyObject(hSession, aliceKeyHandle);
     assert(rv == CKR_OK);
     rv = pF->C_DestroyObject(hSession, secretKeyHandle);
     assert(rv == CKR_OK);

     return  memcmp(secretKeyValue,  ETALON,  sizeof(ETALON))  ?
CKR_FUNCTION_FAILED : CKR_OK;
}
```