ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ
(РОССТАНДАРТ)

Технический комитет 026

«Криптографическая защита информации»

Информационная технология

ТЕХНИЧЕСКАЯ СПЕЦИФИКАЦИЯ

Расширение PKCS#11 для использования российских стандартов ГОСТ Р 34.10-2012 и ГОСТ Р 34.11-2012.

Москва

2016

Содержание

1. Аннотация

Данный документ определяет расширение спецификаций PKCS#11 для использования криптографических алгоритмов **ГОСТ Р 34.10-2012, ГОСТ Р 34.11-2012**, а также алгоритмов, построенных на их основе, а также использования ключей, сертификатов, параметров алгоритмов и других объектов, предназначенных для работы с этими стандартами.

2. Список ссылок

В настоящем документе использованы ссылки на следующие стандарты и рекомендации:

- **ГОСТ Р 34.10-2012** — «Информационные технологии. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи», ГОСТ Р 34.10-2012, Национальный стандарт Российской Федерации, Федеральное агентство по техническому регулированию и метрологии, Стандартинформ, 2012.
- **ГОСТ Р 34.11-2012** — «Информационные технологии. Криптографическая защита информации. Функция хэширования», ГОСТ Р 34.11-2012, Национальный стандарт Российской Федерации, Федеральное агентство по техническому регулированию и метрологии, Стандартинформ, 2012.
- **ТК26АЛГ** — Федеральное агентство по техническому регулированию и метрологии (РОС-СТАНДАРТ), Технический комитет №26, «Рекомендации по стандартизации. криптографическим алгоритмам, сопутствующим применению стандартов ГОСТ Р 34.10-2012 и ГОСТ Р 34.11-2012.
- **ТК26ЭК** — Федеральное агентство по техническому регулированию и метрологии (РОС-СТАНДАРТ), Технический комитет №26, «Рекомендации по заданию параметров эллиптических кривых в соответствии с ГОСТ Р 34.10-2012», Москва, 2014.
- **ТК26ЭДВ —** Федеральное агентство по техническому регулированию и метрологии (РОС-СТАНДАРТ), Технический комитет №26, «Рекомендации по стандартизации. Задание параметров скрученных эллиптических кривых Эдвардса в соответствии с ГОСТ Р 34.10-2012», Москва, 2013
- **RFC 4357** — В. Попов, И. Курепкин, С. Леонтьев, «Дополнительные алгоритмы шифрования для использования с алгоритмами по ГОСТ 28147-89, ГОСТ Р 34.10-94, ГОСТ Р 34.10-2001 и ГОСТ Р 34.11-94» (Popov V., Kurepkin I. and S. Leontiev, Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms, IETF RFC 4357, January 2006).

3. Замечание об использовании численных идентификаторов

До момента включения данного дополнения в официальный профиль PKCS#11 и назначения «стандартных» значений для всех приведенных здесь определений численные значения для них выбираются в соответствии со следующими правилами:

1) признаком нестандартного значения (определяемого производителем) является взведенный старший бит (0x80000000);
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 приложении 1.

4. Тип ключа алгоритма ГОСТ Р 34.10-2012

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 Р 34.10-2012** с длиной закрытого ключа 512 бит.

Такой ключ может использоваться с механизмами

```
CKM_GOSTR3410_512_KEY_PAIR_GEN
CKM_GOSTR3410_512
CKM_GOSTR3410_2012_DERIVE
```

5. Наборы параметров

В качестве параметров алгоритма **ГОСТ Р 34.10-2012** допускается использование следующих параметров:

Для длины закрытого ключа 256 бит:

- Параметры подписи по умолчанию с ключом 256 бит id-GostR3410-2001-CryptoPro-A-ParamSet. OID "1.2.643.2.2.35.1". ASN представление {0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x23, 0x01}. Определён в **RFC 4357**.
- Параметры id-GostR3410-2001-CryptoPro-B-ParamSet.OID "1.2.643.2.2.35.2". ASN представление {0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x23, 0x02}. Определён в **RFC 4357**.
- Параметры id-GostR3410-2001-CryptoPro-C-ParamSet. OID "1.2.643.2.2.35.3". ASN представление {0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x23, 0x03}. Определён в **RFC 4357**.
- Параметры обмена по умолчанию с ключом 256 бит id-GostR3410-2001-CryptoPro-XchA-ParamSet. OID "1.2.643.2.2.36.0". ASN представление {0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x24, 0x00}. Определён в **RFC 4357**.

- Параметры id-GostR3410-2001-CryptoPro-XchB-ParamSet. OID "1.2.643.2.2.36.1". ASN представление {0x06, 0x07, 0x2A, 0x85, 0x03, 0x02, 0x02, 0x24, 0x01}. Определён в **RFC 4357**.
- Параметры с ключом 256 для скрученных эллиптических кривых Эдвардса id-tc26-gost-3410-2012-256-paramSetA. OID "1.2.643.7.1.2.1.1.1". ASN представление { 0x06, 0x09, 0x2A, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01, 0x01, 0x01 }. Определён в **ТК26ЭДВ**.

Для длины закрытого ключа 512 бит:

- Параметры по умолчанию с ключом 512 бит id-tc26-gost-3410-2012-512-paramSetA. OID "1.2.643.7.1.2.1.2.1". ASN представление { 0x06, 0x09, 0x2A, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01, 0x02, 0x01 }. Определён в **ТК26ЭК**.
- Рабочие параметры с ключом 512 id-tc26-gost-3410-2012-512-paramSetB. OID "1.2.643.7.1.2.1.2.2". ASN представление { 0x06, 0x09, 0x2A, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01, 0x02, 0x02 }. Определён в **ТК26ЭК**.
- Параметры с ключом 512 для скрученных эллиптических кривых Эдвардса id-tc26-gost-3410-2012-512-paramSetC. OID "1.2.643.7.1.2.1.2.3". ASN представление { 0x06, 0x09, 0x2A, 0x85, 0x03, 0x07, 0x01, 0x02, 0x01, 0x02, 0x03 }. Определён в **ТК26ЭДВ**.

Набор параметров, идентифицируемый OID-ом 1.2.643.7.1.2.1.2.1, является набором по умолчанию для закрытого ключа длины 512 бит. Если в параметрах механизма явно не заданы параметры, используется этот набор.

В объектах ключей и шаблонах, которые передаются в функции, в качестве значений атрибутов CKA_GOSTR3410_PARAMS, эллиптические параметры представляются в виде ASN закодированного OID.

6. 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 ASN закодированный OID, один из определенных в предыдущем разделе. Для ключей CKK_GOSTR3410 длины 256 бит 6 возможных значений, для CKK_GOSTR3410_512 длины 512 бит 3 значения.

Атрибут CKA_VALUE может отсутствовать или быть недоступным.

Пример использования объектов типа CKO_DOMAIN_PARAMETERS для определения возможности использования набора эллиптических параметров приведен в приложении 2.

7. Механизмы хэширования

Список механизмов хэширования расширяется для использования нового российского стандарта ГОСТ Р 34.11-2012. Для двух функций хэширования, вводятся два новых механизма.

Стандарт ГОСТ Р 34.11-2012 определяет две функции хэширования с длинами значений 256 и 512 бит. Для использования этих функций в PKCS#11 вводятся два механизма

```
#define CKM_GOSTR3411_2012_256 (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x012)
#define CKM_GOSTR3411_2012_512 (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x013)
```

с длинами значений 32 и 64 байта соответственно. Эти механизмы не используют параметры.

Используется в C_DigestInit.

Примеры использования механизмов CKM_GOSTR3411_2012_512 и CKM_GOSTR3411_2012_256 находятся в приложениях 3 и 4 соответственно.

8. Механизмы hmac

Определяются две функции hmac, базирующиеся на основании функций хэширования ГОСТ Р 34.11-2012 и определении hmac функции в rfc 2104 http://tools.ietf.org/html/rfc2104

Эти функции соответствуют функциям, описанным в документе **ТК26АЛГ** раздел 5.1.

```
#define CKM_GOSTR3411_2012_256_HMAC (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x014)
#define CKM_GOSTR3411_2012_512_HMAC (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x015)
```

CKM_GOSTR3411_2012_256_HMAC со значениями B = 64, L = 32 с длиной значения 32 байта

и CKM_GOSTR3411_2012_512_HMAC со значениями B = 64, L = 64 с длиной значения 64 байта соответственно. Эти механизмы не используют параметры.

Совместно с этими механизмами допускается использование ключей типа или CKK_GOST28147 или CKK_GENERIC_SECRET.

Используется в C_SignInit

Примеры использования механизмов CKM_GOSTR3411_2012_512_HMAC и CKM_GOSTR3411_2012_256_HMAC находятся в приложениях 5 и 6 соответственно.

9. Механизмы PRF

При реализации протокола TLS с российскими алгоритмами в соответствии с документом [ТК26TLS] «ИСПОЛЬЗОВАНИЕ НАБОРОВ АЛГОРИТМОВ ШИФРОВАНИЯ НА ОСНОВЕ ГОСТ 28147-89 ДЛЯ ПРОТОКОЛА БЕЗОПАСНОСТИ ТРАНСПОРТНОГО УРОВНЯ (TLS)» используются алгоритмы PRF_TLS_GOSTR3411_2012_256 и PRF_TLS_GOSTR3411_2012_512, основанные на стандарте хэширования ГОСТ Р 34.11-2012 и описанные в документе [ТК26АЛГ] «ИСПОЛЬЗОВАНИЕ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ, СОПУТСТВУЮЩИХ ПРИМЕНЕНИЮ

СТАНДАРТОВ ГОСТ Р 34.10-2012 И ГОСТ Р 34.11-2012» раздел 5.2.1 «Псевдослучайные функции протокола TLS» и соответствующие описанию RFC7836 раздел 4.2.1. «PRFs for the TLS Protocol»

Для получения значений псевдослучайных функций (PRF) в соответствии со стандартом PKCS#11(пункт 6.25 в версии 2.30) используется функция C_Derive. Для использования функций PRF_TLS_GOSTR3411_2012_256 и PRF_TLS_GOSTR3411_2012_512 введены механизмы CKM_TLS_GOST_PRF_2012_256 и CKM_TLS_GOST_PRF_2012_512 соответственно, которые определенны

```
#define CKM_TLS_GOST_PRF_2012_256    (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x016)
#define CKM_TLS_GOST_PRF_2012_512    (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x017)
```

Для задания параметров механизма используется структура CK_TLS_PRF_PARAMS. Функция C_Derive 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 NULL_PTR.

Совместно с этими механизмами допускается использование ключей типа или CKK_GOST28147 или CKK_GENERIC_SECRET.

Примеры использования алгоритма prf, базирующегося на основании функции хэширования ГОСТ Р 34.11-2012 длинами 512и 256 бит находятся в приложениях 7 и 8 соответственно.

10. Использование алгоритма хэширования ГОСТ Р 34.11-2012 в механизме генерации ключей PKCS #5 PBKDF2

Генерация ключей по алгоритму PKCS #5 PBKDF2 осуществляется в соответствии со стандартом PKCS#11(пункт 2.26 “PKCS #5 and PKCS #5-style password-based encryption (PBE)”в версии 2.40). Функции C_GenerateKey вызываются с механизмом CKM_PKCS5_PBKD2. В качестве параметра механизма используется структура CK_PKCS5_PBKD2_PARAMS2.

При этом для использования алгоритма хэширования ГОСТ Р 34.11-2012 в поле

```
CK_PKCS5_PBKD2_PSEUDO_RANDOM_FUNCTION_TYPE prf;
```

структуры CK_PKCS5_PBKD2_PARAMS2 должно указываться значение

CKP_PKCS5_PBKD2_HMAC_GOSTR3411_2012_512

Это значение определено следующим образом

```
#define CKP_PKCS5_PBKD2_HMAC_GOSTR3411_2012_512
(CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x003)
```

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

Пример создания ключа из пароля находится в приложении 9.

11. Механизм создания ключей

Для создания ключей длины 512 бит для стандарта ГОСТ Р 34.10-2012 вводится механизм

```
CKM_GOSTR3410_512_KEY_PAIR_GEN
#define CKM_GOSTR3410_512_KEY_PAIR_GEN (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x005)
```

Используется для `C_GenerateKeyPair`.

Механизм не использует параметров. Набор эллиптических параметров задается в шаблонах ключей в виде ASN закодированного OID, как описано в разделе «Наборы параметров». Если параметры не указаны, используется набор по умолчанию.

Пример генерации ключевой пары алгоритма ГОСТ Р 34.10-2012 длиной 512 бит находится в приложении 10.

12. Механизм создания открытого ключа

Механизм создания открытого ключа из закрытого.

```
#define CKM_GOSTR3410_PUBLIC_KEY_DERIVE (CK_VENDOR_PKCS11_RU_TEAM_TC26
|0x00A)
```

Используется в `C_DeriveKey`

Используются параметры, определенные в закрытом ключе, явно не задаются.

Пример создания открытого ключа из закрытого находится в приложении 11.

13. Механизмы для подписи / проверки

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

```
#define CKM_GOSTR3410_512 (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x006)
```

Совместный алгоритм хэширования CKM_GOSTR3411_2012_256 и подписи на ключе длиной 256 бит.

```
#define CKM_GOSTR3410_WITH_GOSTR3411_2012_256
(CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x008)
```

Совместный алгоритм хэширования CKM_GOSTR3411_2012_512 и подписи на ключе длиной 512 бит.

```
#define CKM_GOSTR3410_WITH_GOSTR3411_2012_512 (CK_VENDOR_PKCS11_RU_TEAM_TCK26
|0x009)
```

Используется в `C_SignInit` и `C_VerifyInit`, параметры используются из ключа и явно в алгоритме не задаются.

Пример использования механизма подписи и проверки подписи находится в приложениях 12 и 13.

14. Механизм согласования ключей.

Для реализации алгоритма Диффи-Хеллмана для ключей ГОСТ Р 34.10-2012 вводится механизм выработки производного ключа CKM_GOSTR3410_2012_DERIVE.

```
#define CKM_GOSTR3410_2012_DERIVE (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x007)
```

Является алгоритмом согласования ключей VKO GOST R 34.10-2012, на основе ГОСТ Р 34.11-2012 и используется для согласования ключей ГОСТ Р34.10-2012 (256 и 512 бит) и предназначен для получения ключевого материала длины 256 бит, использующегося в криптографических протоколах.
Механизм соответствует алгоритму, описанному в документе **ТК26АЛГ** раздел 5.3.

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 (little-endian, т.е. младшие байты сначала) представляют собой значение KDF. Значение определяет механизм диверсификации, один из следующих:
  - CKD_NULL - нет диверсификации
  - CKM_KDF_4357
  - CKD_CPDIVERSIFY_KDF или CKM_KDF_GOSTR3411_2012_256

- 4 байта (little-endian) задают длину открытого ключа в байтах. (64 либо 128)

- открытый ключ, длина которого определена предыдущим полем
(64 (либо 128)-байтовый вектор координаты точки X и Y – два вектора длиной по 32(либо 64) байта в little-endian)

- 4 байта (little-endian) задают длину UKM (от 8 байт)

- UKM (n-байтовый вектор в little-endian) , длина определена выше.

(Константа CKD_NULL равна 1, а CKD_CPDIVERSIFY_KDF равна 9 в PKCS#11 v2.30 draft)

Используется в `C_DeriveKey`

Пример использования механизма подписи и проверки подписи находится в приложениях 14 и 15.

Приложение 1. Заголовочный файл со списком используемых идентификаторов.

```
#define NSSCK_VENDOR_PKCS11_RU_TEAM 0xD4321000 /* 0x80000000 | 0x54321000 */
#define CK_VENDOR_PKCS11_RU_TEAM_TC26           NSSCK_VENDOR_PKCS11_RU_TEAM

/* GOST KEY TYPES */
#define CKK_GOSTR3410_256                       CKK_GOSTR3410
#define CKK_GOSTR3410_512                       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x003)
#define CKK_KUZNECHIK                           (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x004)

/* GOST OBJECT ATTRIBUTES */
#define CKA_GOSTR3410_256PARAMS                 CKA_GOSTR3410PARAMS

/* PKCS #5 PRF Functions */
#define CKP_PKCS5_PBKD2_HMAC_GOSTR3411_2012_512 (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x003)

/* GOST MECHANISMS */
#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 |0x005)
#define CKM_GOSTR3410_256                       CKM_GOSTR3410
#define CKM_GOSTR3410_512                       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x006)
#define CKM_GOSTR3410_2012_DERIVE               (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x007)
#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 |0x008)
#define CKM_GOSTR3410_WITH_GOSTR3411_2012_512   (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x009)
#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 |0x00A)

#define CKM_GOSTR3411_94                        CKM_GOSTR3411
#define CKM_GOSTR3411_2012_256                  (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x012)
#define CKM_GOSTR3411_2012_512                  (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x013)
#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 |0x014)
#define CKM_GOSTR3411_2012_512_HMAC             (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x015)
#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 |0x016)
#define CKM_TLS_GOST_PRF_2012_512               (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x017)
#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 |0x018)

#define CKM_KUZNECHIK_KEY_GEN                   (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x019)
#define CKM_KUZNECHIK_ECB                       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x01A)
#define CKM_KUZNECHIK_CTR                       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x01B)
#define CKM_KUZNECHIK_CFB                       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x01C)
#define CKM_KUZNECHIK_OFB                       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x01D)
#define CKM_KUZNECHIK_CBC                       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x01E)
#define CKM_KUZNECHIK_MAC                       (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x01F)

#define CKM_MAGMA_ECB                           CKM_GOST28147_ECB
#define CKM_MAGMA_CTR                           (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x020)
#define CKM_MAGMA_CFB                           (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x021)
#define CKM_MAGMA_OFB                           (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x022)
#define CKM_MAGMA_CBC                           (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x023)
#define CKM_MAGMA_MAC                           (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x024)

#define CKM_KDF_4357                            (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x025)
#define CKM_KDF_GOSTR3411_2012_256              (CK_VENDOR_PKCS11_RU_TEAM_TC26 |0x026)
```

Приложение 2. Пример использования объектов типа CKO_DOMAIN_PARAMETERS для определения возможности использования набора эллиптических параметров.

```
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}; // "1.2.643.2.2.7.1.2.1.2.1"
     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 \"1.2.643.2.2.7.1.2.1.2.1\"
supported.\n");
               break;
          }
     }
     return rv;
}
```

Приложение 3. 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 4. 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 5. 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 6. 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 7. 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 8. 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 9. Примеры использования алгоритма хэширования ГОСТ Р 34.11-2012 в механизме генерации ключей PKCS #5 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 10. 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_EXTRACTABLE,        &bTrue,             sizeof(bTrue) },
          {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 11. 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 12. 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 13. 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 14. Пример использования механизма CKM_GOSTR3410_2012_DERIVE 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 15. Пример использования механизма CKM_GOSTR3410_2012_DERIVE 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;
}
```