Враппер Java LirPKCS11
Руководство программиста

ООО "ЛИССИ-Крипто"

30 августа 2011 г.

Оглавление

1. Введение

Платформа Java определяет набор программных интерфейсов для выполнения криптографических операций. Эти интерфейсы известны как Java Cryptography Architecture (JCA) и Java Cryptography Extension (JCE), и определены в разделе Security на домашней странице J2SE 6.

Криптографические интерфейсы базируются на провайдерах. Это означает, что приложения общаются с прикладными программными интерфейсами (API), а реальные криптографические операции выполняются в конфигурированных провайдерах, которые наследуются от набора интерфейсов сервис-провайдера (SPI). Данная архитектура поддерживает различные реализации провайдеров. Некоторые провайдеры могут выполнять криптографические операции программно; другие могут выполнять эти операции в аппаратном токене.

Стандарт интерфейса криптографических токенов, PKCS#11, был создан RSA Security и определяет родные программные интерфейсы для криптографических токенов.

Для обеспечения интеграции токенов PKCS#11 на платформе Java имеется штатный криптографический провайдер SunPKCS11. Данный криптопровайдер работает с библиотеками PKCS#11 с помощью интерфейсной прослойки – враппера. К сожалению, данный криптопровайдер не поддерживает работу с российскими криптографическими алгоритмами. Компания "ЛИССИ-Крипто"разработала специальную версию провайдера LirPKCS11 для поддержки российских криптографических алгоритмов.

Приложения Java могут использовать методы враппера и без использования интерфейсов JCA, если приложениям достаточно интерфейсов PKCS#11 для реализации своего функционала. Компания "ЛИССИ-Крипто"разработала новый пакет враппера ru.lissi.security.pkcs11.wrapper с поддержкой российской криптографии, классы которого входят в состав провайдера LirPKCS11. Программирование на более высоком уровне с использованием интерфейсов JCA детально описано в другом документе: "Криптографический сервис-провайдер Java LirPKCS11. Руководство программиста".

2. Враппер LirPKCS11

Враппер LirPKCS11 сам не реализует никакие криптографические алгоритмы. Вместо этого, он действует как мост между Java, с одной стороны, и родным криптографическим PKCS#11 API, с другой стороны, транслируя вызовы и соглашения между ними. Это означает, что приложения Java могут воспользоваться алгоритмами, предоставляемыми нижележащими реализациями PKCS#11, подключая различные криптографические устройства:

- Криптографические токены
- Криптографические смарткарты
- Аппаратные криптографические ускорители
- Высокопроизводительные программные реализации

Заметим, что Java SE только обеспечивает доступ к реализациям PKCS#11, но сама не включает реализацию PKCS#11. Однако, криптографические устройства часто поставляются с программным обеспечением, включающим реализацию PKCS#11, которую нужно установить и конфигурировать в соответствии с инструкциями производителя.

2.1. Требования

Враппер LirPKCS11 поддерживается в Solaris (SPARC и x86), в Linux (x86) и в Windows(x86 и x64) в 32-битных и 64-битных процессах Java.

Враппер LirPKCS11 требует установленной в системе библиотеки PKCS#11 v2.30 или выше. Данная библиотека должна быть выполненной в форме разделяемой библиотеки (.so в Solaris или в Linux) или динамически загружаемой библиотеки (.dll в Windows). Пожалуйста, обратитесь к документации поставщика, чтобы выяснить, включает ли ваше криптографическое устройство такую библиотеку PKCS#11, как ее конфигурировать и как называется файл библиотеки.

Враппер LirPKCS11 поддерживает множество алгоритмов, предполагая, что нижележащая реализация PKCS#11 предоставляет их. Алгоритмы и соответствующие механизмы PKCS#11 перечислены в таблице в Приложении A.

2.2. Конфигурация

Враппер PKCS#11 реализован пакетом ru.lissi.security.pkcs11.wrapper. Для прямого использования враппера нужно сначала создать экземпляр объекта с указани-

ем пути к библиотеке PKCS#11. Файл провайдера LirPKCS11.jar нужно предварительно разместить в папке lib/ext, а JNI-библиотеку (lirj2pkcs11.dll в Windows или lirj2pkcs11.so в Solaris и Linux) – в папке bin используемой JRE.

3. Примеры программ

Заметим, что библиотеки PKCS#11 и соответствующие токены различных производителей могут поддерживать весьма ограниченный набор криптографических механизмов и прочих возможностей, определенных стандартом. Как правило, аппаратные токены применяются только для выполнения операций с использованием хранимого на нем закрытого ключа. Поэтому не все приведенные ниже примеры смогут работать со всеми библиотеками.

Библиотека LCCryptoki фирмы "ЛИССИ-Крипто"работает с программными токенами и обеспечивает расширенную поддержку стандарта PKCS#11 с учетом спецификаций Рабочей группы ТК 26 для российской криптографии. Все приведеные ниже примеры работают с библиотекой LCCryptoki.

3.1. Получение информации

```
package pkcs11Test.pkcs11Info;

import ru.lissi.security.pkcs11.wrapper.*;

public class PKCS11Info {
    public static void main(String[] args) {
System.out.println("Info test");
try {
            PKCS11 pkcs11Wrapper;
            long[] slotList;
            CK_SLOT_INFO slotInfo;
            CK_TOKEN_INFO tokenInfo;

            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();

            // Получаем экзепляр класса
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper =
                PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);

            // Для Java 1.5 используется другой вызов
```

```
/*
            pkcs11Wrapper =
                PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
*/

            // Получаем сведения о библиотеке
            CK_INFO info = new CK_INFO();
            info = pkcs11Wrapper.C_GetInfo();

            System.out.println("Library version: " + info.libraryVersion);
            System.out.println("Manufacturer: "
                        + new String(info.manufacturerID));
            System.out.format("Flags: 0x%H\n", info.flags);
            System.out.println("Library Description: "
                        + new String(info.libraryDescription));
            System.out.println("Cryptoki version: " + info.cryptokiVersion);

            // Получаем список слотов, в которые вставлены токены
            slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим
            if(slotList.length == 0) {
System.out.print("No slots found.");
System.exit(-1);
            }

            for(int i=0; i<slotList.length;i++) {
// Для каждого слота получаем информацию о слоте
                // и информацию о подключенном токене
slotInfo = pkcs11Wrapper.C_GetSlotInfo(slotList[i]);
tokenInfo = pkcs11Wrapper.C_GetTokenInfo(slotList[i]);

System.out.println("Slot #" + slotList[i]);
System.out.println("\tSlot description: "
                        + new String(slotInfo.slotDescription));
System.out.println("\tManufacturer ID: "
                        + new String(slotInfo.manufacturerID));
System.out.print("\tFlags: ");
                System.out.format("Flags: 0x%H\n", slotInfo.flags);
System.out.println("\tHardware version: "
                        + slotInfo.hardwareVersion);
System.out.println("\tFirmware version: "
```

```
                        + slotInfo.firmwareVersion);

System.out.println("\n\tToken:");

System.out.println("\t\tLabel: "
                        + new String(tokenInfo.label));
System.out.println("\t\tManufacturer ID: "
                        + new String(tokenInfo.manufacturerID));
System.out.println("\t\tModel: "
                        + new String(tokenInfo.model));
System.out.println("\t\tSerial Number: "
                        + new String(tokenInfo.serialNumber));
                System.out.format("\t\tFlags: 0x%H\n", tokenInfo.flags);
System.out.println("\t\tMax session count: "
                        + tokenInfo.ulMaxSessionCount);
System.out.println("\t\tSession count: "
                        + tokenInfo.ulSessionCount);
System.out.println("\t\tMax R/W session count: "
                        + tokenInfo.ulMaxRwSessionCount);
System.out.println("\t\tR/W session count: "
                        + tokenInfo.ulRwSessionCount);
System.out.println("\t\tMax Pin length: "
                        + tokenInfo.ulMaxPinLen);
System.out.println("\t\tMin Pin length: "
                        + tokenInfo.ulMinPinLen);
System.out.println("\t\tTotal public memory: "
                        + tokenInfo.ulTotalPublicMemory);
System.out.println("\t\tFree public memory: "
                        + tokenInfo.ulFreePublicMemory);
System.out.println("\t\tTotal private memory: "
                        + tokenInfo.ulTotalPrivateMemory);
System.out.println("\t\tFree private memory: "
                        + tokenInfo.ulFreePrivateMemory);
System.out.println("\t\tHardware version: "
                        + tokenInfo.hardwareVersion);
System.out.println("\t\tFirmware version: "
                        + tokenInfo.firmwareVersion);
            }
            System.out.println("SUCCESS");
}
catch(Exception e) {
            System.out.print(e.getMessage());
}
    }
```

```
}
```

3.2. Генерация случайных байтов

```
package pkcs11Test.pkcs11Random;

import ru.lissi.security.pkcs11.wrapper.*;
import pkcs11Test.utils.*;

public class PKCS11Random {
    private final static char[] Pin =
        {'0', '1', '2', '3', '4', '5', '6', '7'};

    public static void main(String[] args) {
        System.out.println("Random test");
try{
            PKCS11 pkcs11Wrapper;

            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                    new CK_C_INITIALIZE_ARGS();

            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);

            // Для Java 1.5 используется другой вызов
/*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
*/
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.out.print("No slots found.");
System.exit(-1);
            }

            // Открываем сессию.
            long session = pkcs11Wrapper.C_OpenSession(
```

```
                    slotList[0],
                    PKCS11Constants.CKF_SERIAL_SESSION |
                    PKCS11Constants.CKF_RW_SESSION,
                    null, null);

            // Предъявляем ПИН-код пользователя.
            pkcs11Wrapper.C_Login(session, PKCS11Constants.CKU_USER, Pin);

            // Создаем массив для получения
            // последовательности случайных чисел.
            byte[] random = new byte[32];

            pkcs11Wrapper.C_GenerateRandom(session, random);

            utils.hexDump("Random sequence:", random);
            System.out.println("SUCCESS");
}
catch(Exception e){
            System.out.print(e.getMessage());
}
    }
}
```

3.3. Генерация ключевой пары

```
package pkcs11Test.pkcs11CreateKeys;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;

public class PKCS11CreateKeys {
    private final static byte[] STR_CRYPTO_PRO_A = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x23, (byte)0x01};
    private final static byte[] STR_CRYPTO_PRO_HASH1 = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01};


    private final static char[] Pin =
        {'0', '1', '2', '3', '4', '5', '6', '7'};

    public static void main(String[] args) {
```

```
        System.out.println("CKM_GOSTR3410_KEY_PAIR_GEN test");
try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();

            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);

            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.out.print("No slots found.");
System.exit(-1);
            }

            // Открываем сессию.
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);

            // Предъявляем ПИН-код пользователя.
            pkcs11Wrapper.C_Login(session, CKU_USER, Pin);

            // Задаем атрибуты закрытого ключа.
            CK_ATTRIBUTE[] privateKeyAttribute = {
                new CK_ATTRIBUTE(CKA_TOKEN, true),
new CK_ATTRIBUTE(CKA_PRIVATE, true),
new CK_ATTRIBUTE(CKA_LABEL,
                 new String("Private key").toCharArray())
            };

            // Задаем атрибуты открытого ключа.
```

```
            CK_ATTRIBUTE[] publicKeyAttribute = {
                new CK_ATTRIBUTE(CKA_TOKEN, true),
new CK_ATTRIBUTE(CKA_PRIVATE, false),
new CK_ATTRIBUTE(CKA_MODIFIABLE, false),
new CK_ATTRIBUTE(CKA_LABEL,
                    new String("Public key").toCharArray()),
new CK_ATTRIBUTE(CKA_GOSTR3410PARAMS, STR_CRYPTO_PRO_A),
new CK_ATTRIBUTE(CKA_GOSTR3411PARAMS, STR_CRYPTO_PRO_HASH1)
            };

            // Создаем ключевую пару.
            CK_MECHANISM mech = new CK_MECHANISM(CKM_GOSTR3410_KEY_PAIR_GEN);
            long[] keys = pkcs11Wrapper.C_GenerateKeyPair(
                session, mech, publicKeyAttribute, privateKeyAttribute);
            // Удаляем открытый ключ с токена
            pkcs11Wrapper.C_DestroyObject(session, keys[0]);
            // Удаляем закрытый ключ с токена
            pkcs11Wrapper.C_DestroyObject(session, keys[1]);

            pkcs11Wrapper.C_CloseSession(session);

            System.out.println("SUCCESS");
        }
catch(Exception e){
            System.out.print(e.getMessage());
}
    }
}
```

3.4. ЭЦП

```
package pkcs11Test.pkcs11SignAndVerify;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import pkcs11Test.utils.*;

public class PKCS11SignAndVerify {
    private final static byte[] STR_CRYPTO_PRO_A = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x23, (byte)0x01
    };
    private final static byte[] STR_CRYPTO_PRO_HASH1 = {
```

```
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01
    };
    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };
    private final static byte[] message = {
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4
    };

    public static void main(String[] args) {
System.out.println("CKM_GOSTR3410 test");
try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.out.print("No slots found.");
System.exit(-1);
            }

            // Открываем сессию.
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);
```

```
// Предъявляем ПИН-код пользователя.
pkcs11Wrapper.C_Login(session, CKU_USER, Pin);

// Задаем атрибуты закрытого ключа.
CK_ATTRIBUTE[] privateKeyAttribute = {
    new CK_ATTRIBUTE(CKA_TOKEN, true),
    new CK_ATTRIBUTE(CKA_PRIVATE, true),
    new CK_ATTRIBUTE(CKA_LABEL,
        new String("Private key").toCharArray())
};

// Задаем атрибуты открытого ключа.
CK_ATTRIBUTE[] publicKeyAttribute = {
    new CK_ATTRIBUTE(CKA_TOKEN, true),
    new CK_ATTRIBUTE(CKA_PRIVATE, false),
    new CK_ATTRIBUTE(CKA_MODIFIABLE, false),
    new CK_ATTRIBUTE(CKA_LABEL,
        new String("Public key").toCharArray()),
    new CK_ATTRIBUTE(CKA_GOSTR3410PARAMS, STR_CRYPTO_PRO_A),
    new CK_ATTRIBUTE(CKA_GOSTR3411PARAMS, STR_CRYPTO_PRO_HASH1)
};

// Атрибут значения открытого ключа
CK_ATTRIBUTE pubKeyValueAttribute =
    new CK_ATTRIBUTE(CKA_VALUE);
CK_ATTRIBUTE[] pubKeyValueAttributes = { pubKeyValueAttribute };

// Создаем ключевую пару.
long[] keys = pkcs11Wrapper.C_GenerateKeyPair(
    session,
    new CK_MECHANISM(CKM_GOSTR3410_KEY_PAIR_GEN),
    publicKeyAttribute,
    privateKeyAttribute);

// Получаем значение открытого ключа
// для проверки работы функции
pkcs11Wrapper.C_GetAttributeValue(
    session, keys[0], pubKeyValueAttributes);
// Значение открытого ключа
byte[] pubKeyValue = pubKeyValueAttributes[0].getByteArray();
utils.hexDump("Public key value:", pubKeyValue);

// Инициализируем операцию подписи.
pkcs11Wrapper.C_SignInit(
```

```
                    session, new CK_MECHANISM(CKM_GOSTR3410), keys[1]);

            // Подписываем массив message.
            byte[] signature = pkcs11Wrapper.C_Sign(session, message);
            utils.hexDump("Signature value:", signature);

            // Инициализируем операцию проверки подписи.
            pkcs11Wrapper.C_VerifyInit(
                session, new CK_MECHANISM(CKM_GOSTR3410), keys[0]);

            //Проверяем подпись.
            pkcs11Wrapper.C_Verify(session, message, signature);

            System.out.println(
                "Signature created and verified successfully");

            pkcs11Wrapper.C_DestroyObject(session, keys[0]);
            pkcs11Wrapper.C_DestroyObject(session, keys[1]);
            System.out.println("SUCCESS");
}
catch(Exception e){
            System.out.print(e.getMessage());
}
    }
}
```

3.5. Генерация ключей согласования

```
package pkcs11Test.pkcs11DH;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import java.util.Arrays;

public class PKCS11DH {
    private final static byte[] STR_CRYPTO_PRO_A = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x23, (byte)0x01
    };
    private final static byte[] STR_CRYPTO_PRO_HASH1 = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01
    };
```

```
    private final static byte[] STR_CRYPTO_PRO_CIPHER_A = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1f, (byte)0x01
    };
    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };

    public static void main(String[] args) {
        System.out.println("CKM_GOSTR3410_DERIVE test");
try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.out.print("No slots found.");
System.exit(-1);
            }

            // Открываем сессию отправителя.
            long sendSession = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);
            // Открываем сессию получателя.
            long recpSession = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);
            // Предъявляем ПИН-код пользователя.
```

```
pkcs11Wrapper.C_Login(sendSession, CKU_USER, Pin);
// Атрибуты закрытого ключа.
CK_ATTRIBUTE[] privateKeyAttribute = {
    new CK_ATTRIBUTE(CKA_TOKEN, true),
    new CK_ATTRIBUTE(CKA_PRIVATE, true),
    new CK_ATTRIBUTE(CKA_LABEL,
        new String("Private key").toCharArray())
};
// Атрибуты открытого ключа.
CK_ATTRIBUTE[] publicKeyAttribute = {
    new CK_ATTRIBUTE(CKA_TOKEN, true),
    new CK_ATTRIBUTE(CKA_PRIVATE, false),
    new CK_ATTRIBUTE(CKA_MODIFIABLE, false),
    new CK_ATTRIBUTE(CKA_LABEL,
        new String("Public key").toCharArray()),
    new CK_ATTRIBUTE(CKA_GOSTR3410PARAMS,
        STR_CRYPTO_PRO_A),
    new CK_ATTRIBUTE(CKA_GOSTR3411PARAMS,
        STR_CRYPTO_PRO_HASH1)
};
// Атрибуты ключа Диффи-Хеллмана
CK_ATTRIBUTE[] derivedKeyAttribute = {
    new CK_ATTRIBUTE(CKA_TOKEN, false),
    new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
    new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOST28147),
    new CK_ATTRIBUTE(CKA_PRIVATE, false),
    new CK_ATTRIBUTE(CKA_ENCRYPT, true),
    new CK_ATTRIBUTE(CKA_DECRYPT, true),
    new CK_ATTRIBUTE(CKA_WRAP, true),
    new CK_ATTRIBUTE(CKA_UNWRAP, true),
    new CK_ATTRIBUTE(CKA_GOST28147PARAMS,
        STR_CRYPTO_PRO_CIPHER_A)
};
// Параметры для вывода ключа Диффи-Хеллмана
byte[] UKM = {
    (byte)0x28,(byte)0xaf,(byte)0xc5,(byte)0x50,
    (byte)0x9d,(byte)0x0c,(byte)0x74,(byte)0xb3
};
CK_GOSTR3410_DERIVE_PARAMS deriveParams =
        new CK_GOSTR3410_DERIVE_PARAMS(0, null, null);
// Атрибут значения открытого ключа отправителя
CK_ATTRIBUTE[] sendPubKeyValueAttribute = {
    new CK_ATTRIBUTE(PKCS11Constants.CKA_VALUE)
};
```

```
// Атрибут значения открытого ключа получателя
CK_ATTRIBUTE[] recpPubKeyValueAttribute = {
    new CK_ATTRIBUTE(PKCS11Constants.CKA_VALUE)
};
// Создаем ключевую пару отправителя.
long[] sendKeys = pkcs11Wrapper.C_GenerateKeyPair(
    sendSession,
    new CK_MECHANISM(CKM_GOSTR3410_KEY_PAIR_GEN),
    publicKeyAttribute,
    privateKeyAttribute);
// Получаем значение открытого ключа отправителя.
pkcs11Wrapper.C_GetAttributeValue(
    sendSession, sendKeys[0], sendPubKeyValueAttribute);
// Создаем ключевую пару получателя.
long[] recpKeys = pkcs11Wrapper.C_GenerateKeyPair(
    recpSession,
    new CK_MECHANISM(CKM_GOSTR3410_KEY_PAIR_GEN),
    publicKeyAttribute,
    privateKeyAttribute);
// Значение открытого ключа отправителя
byte[] sendPubKeyValue =
    sendPubKeyValueAttribute[0].getByteArray();
// Получаем значение открытого ключа получателя.
pkcs11Wrapper.C_GetAttributeValue(
    recpSession, recpKeys[0], recpPubKeyValueAttribute);
// Значение открытого ключа получателя
byte[] recpPubKeyValue =
    recpPubKeyValueAttribute[0].getByteArray();
// Атрибут значения ключа Диффи-Хеллмана отправителя
CK_ATTRIBUTE[] sendDHKeyValueAttribute = {
    new CK_ATTRIBUTE(PKCS11Constants.CKA_VALUE)
};
// Атрибут значения ключа Диффи-Хеллмана получателя
CK_ATTRIBUTE[] recpDHKeyValueAttribute = {
    new CK_ATTRIBUTE(PKCS11Constants.CKA_VALUE)
};
// Задаем параметры вывода
// ключа Диффи-Хеллмана отправителя
deriveParams.kdf = CKD_CPDIVERSIFY_KDF;
deriveParams.pPublicData = recpPubKeyValue;
deriveParams.pUKM = UKM;
// Создаем механизм вывода ключа Диффи-Хеллмана
CK_MECHANISM mechDerive =
    new CK_MECHANISM(CKM_GOSTR3410_DERIVE, deriveParams);
```

```
            // Генерируем ключ Диффи-Хеллмана отправителя
            long sendDHKey = pkcs11Wrapper.C_DeriveKey(
                sendSession, mechDerive,
                sendKeys[1], derivedKeyAttribute);
            // Генерируем ключ Диффи-Хеллмана получателя
            deriveParams.pPublicData = sendPubKeyValue;
            long recpDHKey = pkcs11Wrapper.C_DeriveKey(
                recpSession, mechDerive,
                recpKeys[1], derivedKeyAttribute);
            // Получаем значение ключа Диффи-Хеллмана отправителя
            pkcs11Wrapper.C_GetAttributeValue(
                sendSession, sendDHKey, sendDHKeyValueAttribute);
            // Значение ключа Диффи-Хеллмана отправителя
            byte[] sendDHKeyValue =
                sendDHKeyValueAttribute[0].getByteArray();
            // Получаем значение ключа Диффи-Хеллмана получателя
            pkcs11Wrapper.C_GetAttributeValue(
                recpSession, recpDHKey, recpDHKeyValueAttribute);
            // Значение ключа Диффи-Хеллмана получателя
            byte[] recpDHKeyValue =
                recpDHKeyValueAttribute[0].getByteArray();
            // Сравниваем значения ключей Диффи-Хеллмана
            if (!Arrays.equals(sendDHKeyValue, recpDHKeyValue)) {
                System.out.println(
                    "Sender and recipient DH keys are different");
                System.exit(-2);
            }
            System.out.println(
                "Sender and recipient DH keys are equal");
            System.out.println("SUCCESS");
}
catch(Exception e){
            System.out.println(e.getMessage());
}
    }
}
```

3.6. Шифрование ключей

```
package pkcs11Test.pkcs11Wrap;

import ru.lissi.security.pkcs11.wrapper.*;
```

```
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import java.util.Arrays;

public class PKCS11Wrap {
    private final static byte[] STR_CRYPTO_PRO_A =
        {(byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
         (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x23, (byte)0x01};
    private final static byte[] STR_CRYPTO_PRO_HASH1 =
        {(byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
         (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01};
    private final static byte[] STR_CRYPTO_PRO_CIPHER_A =
        {(byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
         (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1f, (byte)0x01};
    private final static char[] Pin =
        {'0', '1', '2', '3', '4', '5', '6', '7'};

    public static void main(String[] args) {
System.out.println("CKM_GOSTR3410_KEY_WRAP test");
        try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);
            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
                System.err.println("No slots found.");
                System.exit(-1);
            }

            // Открываем сессию отправителя.
            long sendSession = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
```

```
    null, null);
// Открываем сессию получателя.
long recpSession = pkcs11Wrapper.C_OpenSession(
    slotList[0],
    CKF_SERIAL_SESSION | CKF_RW_SESSION,
    null, null);
// Предъявляем ПИН-код пользователя.
pkcs11Wrapper.C_Login(sendSession, CKU_USER, Pin);
// Атрибуты закрытого ключа.
CK_ATTRIBUTE[] privateKeyAttribute = {
    new CK_ATTRIBUTE(CKA_TOKEN, true),
    new CK_ATTRIBUTE(CKA_PRIVATE, true),
    new CK_ATTRIBUTE(CKA_LABEL,
            new String("Private key").toCharArray())
};
// Атрибуты открытого ключа.
CK_ATTRIBUTE[] publicKeyAttribute = {
    new CK_ATTRIBUTE(CKA_TOKEN, true),
    new CK_ATTRIBUTE(CKA_PRIVATE, false),
    new CK_ATTRIBUTE(CKA_MODIFIABLE, false),
    new CK_ATTRIBUTE(CKA_LABEL,
        new String("Public key").toCharArray()),
    new CK_ATTRIBUTE(CKA_GOSTR3410PARAMS,
        STR_CRYPTO_PRO_A),
    new CK_ATTRIBUTE(CKA_GOSTR3411PARAMS,
        STR_CRYPTO_PRO_HASH1)
};
// Значение секретного ключа
byte[] send_cipher_key_val = {
    (byte)0xc3,(byte)0x73,(byte)0x0c,(byte)0x5c,
    (byte)0xbc,(byte)0xca,(byte)0xcf,(byte)0x91,
    (byte)0x5a,(byte)0xc2,(byte)0x92,(byte)0x67,
    (byte)0x6f,(byte)0x21,(byte)0xe8,(byte)0xbd,
    (byte)0x4e,(byte)0xf7,(byte)0x53,(byte)0x31,
    (byte)0xd9,(byte)0x40,(byte)0x5e,(byte)0x5f,
    (byte)0x1a,(byte)0x61,(byte)0xdc,(byte)0x31,
    (byte)0x30,(byte)0xa6,(byte)0x50,(byte)0x11
};
// Атрибуты секретного ключа
CK_ATTRIBUTE[] cipher_key_attr = {
    new CK_ATTRIBUTE(CKA_VALUE, send_cipher_key_val),
    new CK_ATTRIBUTE(CKA_SENSITIVE, false),
    new CK_ATTRIBUTE(CKA_EXTRACTABLE, true),
    new CK_ATTRIBUTE(CKA_WRAP, true),
```

```
        new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
        new CK_ATTRIBUTE(CKA_GOST28147PARAMS,
            STR_CRYPTO_PRO_CIPHER_A),
        new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOST28147),
    };
    // Атрибуты распаковываемого секретного ключа
    CK_ATTRIBUTE[] unwrapped_cipher_key_attr = {
        new CK_ATTRIBUTE(CKA_SENSITIVE, false),
        new CK_ATTRIBUTE(CKA_EXTRACTABLE, true),
        new CK_ATTRIBUTE(CKA_WRAP, true),
        new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
        new CK_ATTRIBUTE(CKA_GOST28147PARAMS,
            STR_CRYPTO_PRO_CIPHER_A),
        new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOST28147),
    };
    // Параметры для генерации ключа шифрования
    byte[] ukm = {
        (byte)0x28,(byte)0xaf,(byte)0xc5,(byte)0x50,
        (byte)0x9d,(byte)0x0c,(byte)0x74,(byte)0xb3
    };
    // Создаем объект секретного ключа
    long send_cipher_key = pkcs11Wrapper.C_CreateObject(
            sendSession, cipher_key_attr);
    // Атрибут значения открытого ключа отправителя
    CK_ATTRIBUTE[] sendPubKeyValueAttribute =
        {new CK_ATTRIBUTE(PKCS11Constants.CKA_VALUE)};
    // Атрибут значения открытого ключа получателя
    CK_ATTRIBUTE[] recpPubKeyValueAttribute =
        {new CK_ATTRIBUTE(PKCS11Constants.CKA_VALUE)};
    // Создаем ключевую пару отправителя.
    long[] sendKeys = pkcs11Wrapper.C_GenerateKeyPair(
        sendSession,
        new CK_MECHANISM(CKM_GOSTR3410_KEY_PAIR_GEN),
        publicKeyAttribute,
        privateKeyAttribute);
    // Получаем значение открытого ключа отправителя.
    pkcs11Wrapper.C_GetAttributeValue(sendSession,
        sendKeys[0], sendPubKeyValueAttribute);
    // Создаем ключевую пару получателя.
    long[] recpKeys = pkcs11Wrapper.C_GenerateKeyPair(
        recpSession,
        new CK_MECHANISM(CKM_GOSTR3410_KEY_PAIR_GEN),
        publicKeyAttribute,
        privateKeyAttribute);
```

```
// Значение открытого ключа отправителя
byte[] sendPubKeyValue =
    sendPubKeyValueAttribute[0].getByteArray();
// Получаем значение открытого ключа получателя.
pkcs11Wrapper.C_GetAttributeValue(
    recpSession, recpKeys[0], recpPubKeyValueAttribute);
// Значение открытого ключа получателя
byte[] recpPubKeyValue =
    recpPubKeyValueAttribute[0].getByteArray();
// Атрибуты создания открытого ключа
CK_ATTRIBUTE[] recp_pub_key_attr = {
    new CK_ATTRIBUTE(CKA_CLASS, CKO_PUBLIC_KEY),
    new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOSTR3410),
    new CK_ATTRIBUTE(CKA_GOSTR3410PARAMS,
        STR_CRYPTO_PRO_A),
    new CK_ATTRIBUTE(CKA_GOSTR3411PARAMS,
        STR_CRYPTO_PRO_HASH1),
    new CK_ATTRIBUTE(CKA_VALUE, recpPubKeyValue),
};
// Создаем объект открытого ключа
// получателя в сессии отправителя
long recp_pub_key = pkcs11Wrapper.C_CreateObject(
    sendSession, recp_pub_key_attr);

// Вариант упаковки:
// 1.0 Эфемерная ключевая пара,
// случайное значение UKM - 167 байтов
System.out.println("Ephemeral keypair, random UKM");
CK_GOSTR3410_KEY_WRAP_PARAMS params =
    new CK_GOSTR3410_KEY_WRAP_PARAMS(
        STR_CRYPTO_PRO_CIPHER_A, null, 0);
CK_MECHANISM mech =
    new CK_MECHANISM(CKM_GOSTR3410_KEY_WRAP, params);
byte[] wrapped_key = pkcs11Wrapper.C_WrapKey(
    sendSession, mech, recp_pub_key, send_cipher_key);
System.out.println("Wrapped key length: "
    + wrapped_key.length);

// Вариант распаковки:
// 1. Эфемерная ключевая пара,
// случайное значение UKM - 167 байтов
params.hKey = CK_INVALID_HANDLE;
long recp_cipher_key = pkcs11Wrapper.C_UnwrapKey(
    recpSession, mech, recpKeys[1],
```

```
    wrapped_key, unwrapped_cipher_key_attr);
// Атрибут значения распакованного секретного ключа
CK_ATTRIBUTE[] cipherKeyValueAttribute =
    {new CK_ATTRIBUTE(CKA_VALUE)};
// Получаем значение распакованного ключа
pkcs11Wrapper.C_GetAttributeValue(
    recpSession, recp_cipher_key, cipherKeyValueAttribute);
// Значение ключа Диффи-Хеллмана получателя
byte[] recp_cipher_key_val =
    cipherKeyValueAttribute[0].getByteArray();
// Сравниваем значения исходного и распакованного ключей
if (!Arrays.equals(send_cipher_key_val,
    recp_cipher_key_val)) {
    System.err.println(
        "Sender and recipient keys are different");
    System.exit(-2);
}
System.out.println(
    "Sender and recipient keys are equal");
pkcs11Wrapper.C_DestroyObject(recpSession, recp_cipher_key);

// Вариант упаковки:
// 2. Заданная ключевая пара,
// заданное значение UKM - 65 байтов
System.out.println("Given keypair, given UKM");

params = new CK_GOSTR3410_KEY_WRAP_PARAMS(
    STR_CRYPTO_PRO_CIPHER_A, ukm, sendKeys[1]);
mech = new CK_MECHANISM(CKM_GOSTR3410_KEY_WRAP, params);
wrapped_key = pkcs11Wrapper.C_WrapKey(
    sendSession, mech, recp_pub_key, send_cipher_key);
System.out.println("Wrapped key length: "
    + wrapped_key.length);

// Вариант распаковки:
// 2. Заданная ключевая пара,
// заданное значение UKM - 65 байтов
CK_ATTRIBUTE[] send_pub_key_attr = {
    new CK_ATTRIBUTE(CKA_CLASS, CKO_PUBLIC_KEY),
    new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOSTR3410),
    new CK_ATTRIBUTE(CKA_GOSTR3410PARAMS,
        STR_CRYPTO_PRO_A),
    new CK_ATTRIBUTE(CKA_GOSTR3411PARAMS,
        STR_CRYPTO_PRO_HASH1),
```

```
                new CK_ATTRIBUTE(CKA_VALUE, sendPubKeyValue),
            };
            // Создаем объект открытого ключа
            // отправителя в сессии получателя
            params.hKey = pkcs11Wrapper.C_CreateObject(
                recpSession, send_pub_key_attr);
            recp_cipher_key = pkcs11Wrapper.C_UnwrapKey(
                recpSession, mech, recpKeys[1],
                wrapped_key, unwrapped_cipher_key_attr);
            // Получаем значение распакованного ключа
            pkcs11Wrapper.C_GetAttributeValue(
                recpSession, recp_cipher_key, cipherKeyValueAttribute);
            // Значение ключа Диффи-Хеллмана получателя
            recp_cipher_key_val =
                cipherKeyValueAttribute[0].getByteArray();
            // Сравниваем значения исходного и распакованного ключей
            if (!Arrays.equals(send_cipher_key_val,
                recp_cipher_key_val)) {
                System.err.println(
                    "Sender and recipient keys are different");
                System.exit(-2);
            }
            System.out.println("Sender and recipient keys are equal");
            pkcs11Wrapper.C_DestroyObject(sendSession, sendKeys[0]);
            pkcs11Wrapper.C_DestroyObject(sendSession, sendKeys[1]);
            pkcs11Wrapper.C_DestroyObject(sendSession, send_cipher_key);
            pkcs11Wrapper.C_DestroyObject(sendSession, recp_pub_key);
            pkcs11Wrapper.C_DestroyObject(recpSession, recpKeys[0]);
            pkcs11Wrapper.C_DestroyObject(recpSession, recpKeys[1]);
            pkcs11Wrapper.C_DestroyObject(recpSession, recp_cipher_key);
            pkcs11Wrapper.C_DestroyObject(recpSession, params.hKey);
            System.out.println("SUCCESS");
}
catch(Exception e){
        System.out.println(e.getMessage());
}
   }
}
```

3.7. Дайджест

```
package pkcs11Test.pkcs11Digest;
```

```
import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import pkcs11Test.utils.*;

public class PKCS11Digest {
    private final static char[] Pin =
        {'0', '1', '2', '3', '4', '5', '6', '7'};

    private final static byte[] message = {
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4
    };

    public static void main(String[] args) {
System.out.println("Digest test");
try{
            PKCS11 pkcs11Wrapper;

            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();

            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);

            // Для Java 1.5 используется другой вызов
/*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
*/
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.err.println("No slots found.");
System.exit(-1);
            }

            // Открываем сессию.
```

```
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                PKCS11Constants.CKF_SERIAL_SESSION |
                PKCS11Constants.CKF_RW_SESSION,
                null, null);

            // Предъявляем ПИН-код пользователя.
            pkcs11Wrapper.C_Login(session,
                PKCS11Constants.CKU_USER, Pin);

            // Создадем массив для хеш-последовательности.
            byte[] digest = new byte[32];

            // Вычисляем значение хеш-функции
            // от данных в массиве message.
            pkcs11Wrapper.C_DigestSingle(
                session, new CK_MECHANISM(CKM_GOSTR3411),
                message, 0, 32,digest, 0, 32);

            utils.hexDump("Digest:", digest);

            pkcs11Wrapper.C_CloseSession(session);

            System.out.println("SUCCESS");

}
catch(Exception e){
            System.out.print(e.getMessage());
}
    }
}
```

3.8. HMAC

```
package pkcs11Test.pkcs11HMAC;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import pkcs11Test.utils.*;

public class PKCS11HMAC {
    private final static byte[] STR_CRYPTO_PRO_HASH1 = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
```

```
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01
    };
    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };
    private final static byte[] message = {
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4
    };

    public static void main(String[] args) {
        System.out.println("CKM_GOSTR3411_HMAC test");
try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.err.println("No slots found.");
System.exit(0);
            }

            // Открываем сессию.
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);

            // Предъявляем ПИН-код пользователя.
```

```
            pkcs11Wrapper.C_Login(session, CKU_USER, Pin);
            byte[] keyval_32 = {
                (byte)0x30, (byte)0x31, (byte)0x32, (byte)0x33,
                (byte)0x34, (byte)0x35, (byte)0x36, (byte)0x37,
                (byte)0x38, (byte)0x39, (byte)0x61, (byte)0x62,
                (byte)0x63, (byte)0x64, (byte)0x65, (byte)0x66,
                (byte)0x30, (byte)0x31, (byte)0x32, (byte)0x33,
                (byte)0x34, (byte)0x35, (byte)0x36, (byte)0x37,
                (byte)0x38, (byte)0x39, (byte)0x61, (byte)0x62,
                (byte)0x63, (byte)0x64, (byte)0x65, (byte)0x66
            };
            CK_ATTRIBUTE[] secretKeyAttribute = {
                new CK_ATTRIBUTE(CKA_VALUE, keyval_32),
                new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
                new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GENERIC_SECRET),
new CK_ATTRIBUTE(CKA_SIGN, true),
new CK_ATTRIBUTE(CKA_VERIFY, true)
            };

            // Создаем ключ.
            long key = pkcs11Wrapper.C_CreateObject(
                session, secretKeyAttribute);

            // Инициализируем операцию подписи HMAC.
            pkcs11Wrapper.C_SignInit(session,
                new CK_MECHANISM(CKM_GOSTR3411_HMAC), key);

            // Подписываем массив message.
            byte[] signature = pkcs11Wrapper.C_Sign(session, message);
            utils.hexDump("HMAC value:", signature);

            // Инициализируем операцию проверки подписи HMAC.
            pkcs11Wrapper.C_VerifyInit(session,
                new CK_MECHANISM(CKM_GOSTR3411_HMAC, STR_CRYPTO_PRO_HASH1),
                key);

            //Проверяем подпись HMAC.
            pkcs11Wrapper.C_Verify(session, message, signature);

            System.out.println("HMAC created and verified successfully");
            pkcs11Wrapper.C_DestroyObject(session, key);
            System.out.println("SUCCESS");
}
catch(Exception e){
```

```
            System.out.println(e.getMessage());
        }
    }
}
```

3.9. Симметричное шифрование

```
package pkcs11Test.pkcs11Cipher;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import java.util.Arrays;
import pkcs11Test.utils.*;

public class PKCS11Cipher {
    private final static byte[] STR_CRYPTO_PRO_CIPHER_A = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1f, (byte)0x01
    };
    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };
    private final static byte[] message = {
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4
    };

    public static void main(String[] args) {
System.out.println("CKM_GOST28147 test");
try{
         PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
```

```
                ck_c_initialize_args, false);
            */

            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.err.println("No slots found.");
System.exit(-1);
            }

            // Открываем сессию.
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);

            // Предъявляем ПИН-код пользователя.
            pkcs11Wrapper.C_Login(session, CKU_USER, Pin);

            CK_ATTRIBUTE[] cipherKeyAttribute = {
                new CK_ATTRIBUTE(CKA_TOKEN, false),
new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOST28147),
new CK_ATTRIBUTE(CKA_PRIVATE, false),
new CK_ATTRIBUTE(CKA_MODIFIABLE, false),
new CK_ATTRIBUTE(CKA_ENCRYPT, true),
new CK_ATTRIBUTE(CKA_DECRYPT, true),
new CK_ATTRIBUTE(CKA_SIGN, true),
new CK_ATTRIBUTE(CKA_VERIFY, true),
new CK_ATTRIBUTE(CKA_LABEL,
                    new String("Cipher key").toCharArray()),
new CK_ATTRIBUTE(CKA_GOST28147PARAMS,
                    STR_CRYPTO_PRO_CIPHER_A)
            };
            byte[] ivec = {
                (byte)0x0f, (byte)0xed, (byte)0xcb, (byte)0xa9,
                (byte)0x87, (byte)0x65, (byte)0x43, (byte)0x21
            };

            // Создаем ключ.
            long key = pkcs11Wrapper.C_GenerateKey(session,
                new CK_MECHANISM(CKM_GOST28147_KEY_GEN),
```

```
    cipherKeyAttribute);

// Инициализируем операцию шифрования.
CK_MECHANISM mech = new CK_MECHANISM(CKM_GOST28147, ivec);
pkcs11Wrapper.C_EncryptInit(session, mech, key);

utils.hexDump("Plain text:", message);

// Шифруем массив message.
byte[] encrypted = message.clone();
pkcs11Wrapper.C_Encrypt(session,
    message, 0, message.length,
    encrypted, 0, message.length);

utils.hexDump("Encrypted:", encrypted);

// Инициализируем операцию расшифровки.
pkcs11Wrapper.C_DecryptInit(session, mech, key);

// Расшифровываем.
byte[] decrypted = encrypted.clone();
pkcs11Wrapper.C_Decrypt(session,
    encrypted, 0, encrypted.length,
    decrypted, 0, encrypted.length);

utils.hexDump("Decrypted:", decrypted);

if (decrypted.length != message.length) {
    System.err.println(
        "Wrong decrypted length "+decrypted.length);
    System.exit(-1);
}
if (!Arrays.equals(decrypted, message)) {
    System.err.println("Wrong decryption");
    System.exit(-2);
}

// Инициализируем многошаговую операцию шифрования.
pkcs11Wrapper.C_EncryptInit(session, mech, key);
// Шифруем массив message.
encrypted = message.clone();
pkcs11Wrapper.C_EncryptUpdate(session,
    0, message, 0, 10, 0, encrypted, 0, 10);
pkcs11Wrapper.C_EncryptUpdate(session,
```

```
                0, message, 10, message.length-10,
                0, encrypted, 10, message.length-10);
    pkcs11Wrapper.C_EncryptFinal(session, 0, encrypted, 0, 0);
            utils.hexDump("Encrypted:", encrypted);
            // Инициализируем многошаговую операцию расшифровки.
            pkcs11Wrapper.C_DecryptInit(session, mech, key);
            // Расшифровываем.
            decrypted = encrypted.clone();
            pkcs11Wrapper.C_DecryptUpdate(session,
                0, encrypted, 0, 10,
                0, decrypted, 0, 10);
            pkcs11Wrapper.C_DecryptUpdate(session,
                0, encrypted, 10, encrypted.length-10,
                0, decrypted, 10, encrypted.length-10);
    pkcs11Wrapper.C_DecryptFinal(session, 0, decrypted, 0, 0);
            utils.hexDump("Decrypted:", decrypted);
            if (decrypted.length != message.length) {
                System.err.println("Wrong decrypted length "
                    +decrypted.length);
                System.exit(-1);
            }

            if (!Arrays.equals(decrypted, message)) {
                System.err.println("Wrong decryption");
                System.exit(-2);
            }

            System.out.println(
                "Encrypted and decrypted successfully");
            pkcs11Wrapper.C_DestroyObject(session, key);
            System.out.println("SUCCESS");
}
catch(Exception e){
            System.out.print(e.getMessage());
}
    }
}
```

3.10. Имитовставка

```
package pkcs11Test.pkcs11MAC;

import ru.lissi.security.pkcs11.wrapper.*;
```

```
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import pkcs11Test.utils.*;

public class PKCS11MAC {
    private final static byte[] STR_CRYPTO_PRO_CIPHER_A = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1f, (byte)0x01
    };

    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };

    private final static byte[] message = {
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4,
        0x1, 0x2, 0x3, 0x4, 0x1, 0x2, 0x3, 0x4
    };

    public static void main(String[] args) {
System.out.println("CKM_GOST28147_MAC test");
try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */

            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.err.println("No slots found.");
System.exit(0);
```

```
            }

            // Открываем сессию.
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);

            // Предъявляем ПИН-код пользователя.
            pkcs11Wrapper.C_Login(session, CKU_USER, Pin);

            CK_ATTRIBUTE[] cipherKeyAttribute = {
                new CK_ATTRIBUTE(CKA_TOKEN, false),
new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOST28147),
new CK_ATTRIBUTE(CKA_PRIVATE, false),
new CK_ATTRIBUTE(CKA_MODIFIABLE, false),
new CK_ATTRIBUTE(CKA_ENCRYPT, true),
new CK_ATTRIBUTE(CKA_DECRYPT, true),
new CK_ATTRIBUTE(CKA_SIGN, true),
new CK_ATTRIBUTE(CKA_VERIFY, true),
new CK_ATTRIBUTE(CKA_LABEL,
                    new String("Cipher key").toCharArray()),
new CK_ATTRIBUTE(CKA_GOST28147PARAMS,
                    STR_CRYPTO_PRO_CIPHER_A)
            };

            // Создаем ключ.
            long key = pkcs11Wrapper.C_GenerateKey(session,
                new CK_MECHANISM(CKM_GOST28147_KEY_GEN),
                cipherKeyAttribute);

            // Инициализируем операцию подписи MAC.
            pkcs11Wrapper.C_SignInit(session,
                new CK_MECHANISM(CKM_GOST28147_MAC),
                key);

            // Подписываем массив message.
            byte[] signature = pkcs11Wrapper.C_Sign(session, message);
            utils.hexDump("MAC value:", signature);

            // Инициализируем операцию проверки подписи MAC.
            pkcs11Wrapper.C_VerifyInit(session,
                new CK_MECHANISM(CKM_GOST28147_MAC),
```

```
                key);

            //Проверяем подпись MAC.
            pkcs11Wrapper.C_Verify(session, message, signature);

            System.out.println("MAC created and verified successfully");
            pkcs11Wrapper.C_DestroyObject(session, key);
            System.out.println("SUCCESS");
        }
catch(Exception e){
            System.out.print(e.getMessage());
}
    }
}
```

3.11. PKCS#8

```
package pkcs11Test.pkcs11PKCS8;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import java.util.Arrays;
import pkcs11Test.utils.*;

public class PKCS11PKCS8 {
    private final static byte[] STR_CRYPTO_PRO_A =
        {(byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
         (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x23, (byte)0x01};
    private final static byte[] STR_CRYPTO_PRO_HASH1 =
        {(byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
         (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01};
    private final static byte[] STR_CRYPTO_PRO_CIPHER_A =
        {(byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
         (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1f, (byte)0x01};
    private final static char[] Pin =
            {'0', '1', '2', '3', '4', '5', '6', '7'};

    public static void main(String[] args) {
System.out.println(
            "CKM_PBA_GOST28147_PKCS8_KEY_WRAP " + "" +
            "and CKM_GOSTR3410_PUBLIC_KEY_DERIVE test");
        try{
            PKCS11 pkcs11Wrapper;
```

```
CK_C_INITIALIZE_ARGS ck_c_initialize_args =
        new CK_C_INITIALIZE_ARGS();
// Данный метод был изменен в Java 1.6
pkcs11Wrapper = PKCS11.getInstance(
    "lccryptoki_fs",
    "C_GetFunctionList", ck_c_initialize_args, false);
// Для Java 1.5 используется другой вызов
/*
pkcs11Wrapper = PKCS11.getInstance(
    "lccryptoki_fs",
    ck_c_initialize_args, false);
*/
// Получаем список слотов, в которые вставлены токены.
long[] slotList = pkcs11Wrapper.C_GetSlotList(true);
// Если ни одного токена не вставлено -- выходим.
if(slotList.length == 0) {
    System.err.println("No slots found.");
    System.exit(0);
}

// Открываем сессию отправителя.
long sendSession = pkcs11Wrapper.C_OpenSession(
        slotList[0],
        CKF_SERIAL_SESSION | CKF_RW_SESSION, null, null);
// Открываем сессию получателя.
long recpSession = pkcs11Wrapper.C_OpenSession(
        slotList[0],
        CKF_SERIAL_SESSION | CKF_RW_SESSION, null, null);
// Предъявляем ПИН-код пользователя.
pkcs11Wrapper.C_Login(sendSession, CKU_USER, Pin);

long pub_key = CK_INVALID_HANDLE;
long priv_key = CK_INVALID_HANDLE;
long unw_priv_key = CK_INVALID_HANDLE;
long der_pub_key = CK_INVALID_HANDLE;
byte[] salt = {
    (byte)0x12, (byte)0x34, (byte)0x56, (byte)0x78,
    (byte)0x78, (byte)0x56, (byte)0x34, (byte)0x26
};
long iter = 2048;
byte[] iv = {
    (byte)0x12, (byte)0x34, (byte)0x56, (byte)0x78,
    (byte)0x78, (byte)0x56, (byte)0x34, (byte)0xab
};
```

```
            // Use UTF-8 password
            char[] password = "4444".toCharArray();
            CK_ATTRIBUTE[] pub_template = {
                new CK_ATTRIBUTE(CKA_TOKEN, true),
                new CK_ATTRIBUTE(CKA_PRIVATE, false),
                new CK_ATTRIBUTE(CKA_MODIFIABLE, false),
                new CK_ATTRIBUTE(CKA_GOSTR3410PARAMS,
                    STR_CRYPTO_PRO_A),
                new CK_ATTRIBUTE(CKA_GOSTR3411PARAMS,
                    STR_CRYPTO_PRO_HASH1)
            };
            // Template for token private key generation.
            CK_ATTRIBUTE[] priv_template = {
                new CK_ATTRIBUTE(CKA_TOKEN, true),
                new CK_ATTRIBUTE(CKA_PRIVATE, true),
            };
            // Additional attributes for token
            // unwrapped private key generation.
            CK_ATTRIBUTE[] unw_template = {
new CK_ATTRIBUTE(CKA_CLASS, CKO_PRIVATE_KEY),
new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOSTR3410)
            };
            CK_ATTRIBUTE[] pub_der_template = {
                new CK_ATTRIBUTE(CKA_CLASS, CKO_PUBLIC_KEY),
                new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOSTR3410),
                new CK_ATTRIBUTE(CKA_GOSTR3410PARAMS,
                    STR_CRYPTO_PRO_A),
                new CK_ATTRIBUTE(CKA_GOSTR3411PARAMS,
                    STR_CRYPTO_PRO_HASH1)
            };
            byte[] pr_key_bag = {
// 0
    (byte)0x30, (byte)0x81, (byte)0xa7,
// 3
        (byte)0x30, (byte)0x5c,
// 5
             (byte)0x06, (byte)0x09, (byte)0x2a, (byte)0x86,
             (byte)0x48, (byte)0x86, (byte)0xf7, (byte)0x0d,
             (byte)0x01, (byte)0x05, (byte)0x0d,
// 16
                (byte)0x30, (byte)0x4f,
// 18
                    (byte)0x30, (byte)0x2e,
// 20
```

```
                        (byte)0x06, (byte)0x09, (byte)0x2a, (byte)0x86,
                        (byte)0x48, (byte)0x86, (byte)0xf7, (byte)0x0d,
                        (byte)0x01, (byte)0x05, (byte)0x0c,
// 31
                        (byte)0x30, (byte)0x21,
// 33
                            (byte)0x04, (byte)0x08,
// 35 - salt
                              (byte)0x12, (byte)0x34, (byte)0x56, (byte)0x78,
                              (byte)0x78, (byte)0x56, (byte)0x34, (byte)0x26,
// 43
                            (byte)0x02, (byte)0x02,
// 45 - iter (2048)
                              (byte)0x08, (byte)0x00,
// 47
                            (byte)0x30, (byte)0x11,
// 49
                              (byte)0x06, (byte)0x06, (byte)0x2a, (byte)0x85,
                              (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x0a,
// 57 - oid_3411_par
                              (byte)0x06, (byte)0x07, (byte)0x2a, (byte)0x85,
                              (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e,
                              (byte)0x01,
// 66
                        (byte)0x30, (byte)0x1d,
// 68
                            (byte)0x06, (byte)0x06, (byte)0x2a, (byte)0x85,
                            (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x15,
// 76
                            (byte)0x30, (byte)0x13,
// 78
                              (byte)0x04, (byte)0x08,
// 80 - iv
                                (byte)0x12, (byte)0x34, (byte)0x56, (byte)0x78,
                                (byte)0x78, (byte)0x56, (byte)0x34, (byte)0xab,
// 88 - oid_28147_par
                              (byte)0x06, (byte)0x07, (byte)0x2a, (byte)0x85,
                              (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1f,
                              (byte)0x01,
// 97
                (byte)0x04, (byte)0x47,
// 99 - encr_pr_key
                    (byte)0x46, (byte)0x5c, (byte)0xc8, (byte)0x21,
                    (byte)0xcd, (byte)0xcc, (byte)0xb8, (byte)0x99,
```

```
                    (byte)0x53, (byte)0xa7, (byte)0x72, (byte)0xda,
                    (byte)0x20, (byte)0x0f, (byte)0x3d, (byte)0xd5,
                    (byte)0xab, (byte)0x59, (byte)0x11, (byte)0x6f,
                    (byte)0x4f, (byte)0x8a, (byte)0x75, (byte)0x9b,
                    (byte)0xf4, (byte)0xd1, (byte)0x91, (byte)0x7e,
                    (byte)0x9d, (byte)0x2f, (byte)0x79, (byte)0xab,
                    (byte)0x95, (byte)0xb8, (byte)0x54, (byte)0xe2,
                    (byte)0x5b, (byte)0x21, (byte)0x61, (byte)0xa8,
                    (byte)0xe7, (byte)0x2b, (byte)0x58, (byte)0x5b,
                    (byte)0xe2, (byte)0x10, (byte)0xd0, (byte)0xd8,
                    (byte)0xbb, (byte)0xd4, (byte)0x04, (byte)0xdc,
                    (byte)0x06, (byte)0x4c, (byte)0x60, (byte)0x14,
                    (byte)0x60, (byte)0x12, (byte)0xd1, (byte)0xf1,
                    (byte)0xb2, (byte)0xbf, (byte)0x39, (byte)0xf7,
                    (byte)0xc8, (byte)0x35, (byte)0x16, (byte)0xc4,
                    (byte)0xea, (byte)0xe9, (byte)0xb6
// 169
    };
            CK_MECHANISM mechanism_gen =
                new CK_MECHANISM(CKM_GOSTR3410_KEY_PAIR_GEN);
            long keys[] = pkcs11Wrapper.C_GenerateKeyPair(
                sendSession, mechanism_gen,
                pub_template,
priv_template);
            pub_key = keys[0];
            priv_key = keys[1];
            CK_GOST28147_PKCS8_KEY_WRAP_PARAMS params =
                new CK_GOST28147_PKCS8_KEY_WRAP_PARAMS(
                    password, STR_CRYPTO_PRO_HASH1, salt, iter,
                    STR_CRYPTO_PRO_CIPHER_A, iv);
            CK_MECHANISM mechanism =
                new CK_MECHANISM(CKM_GOST28147_PKCS8_KEY_WRAP, params);
            byte[] value = pkcs11Wrapper.C_WrapKey(
                sendSession, mechanism, CK_INVALID_HANDLE, priv_key);
            utils.hexDump("PKCS#8:", value);
            if (value.length != pr_key_bag.length) {
                System.err.println("Invalid length: " + value.length);
                System.exit(-1);
            }

            byte[] val_99 = Arrays.copyOfRange(value, 0, 99);
            utils.hexDump("val_99:", val_99);
            byte[] pr_key_bag_99 = Arrays.copyOfRange(pr_key_bag, 0, 99);
            utils.hexDump("pr_key_bag_99:", pr_key_bag_99);
```

```
            if (!Arrays.equals(val_99, pr_key_bag_99)) {
                System.err.println("Invalid PKCS#8 value");
                System.exit(-2);
            }

            mechanism.pParameter = password;
            unw_priv_key = pkcs11Wrapper.C_UnwrapKey(
                recpSession, mechanism, 0, value, unw_template);
            // Заодно проверим механизм генерации
            // открытого ключа по закрытому
            CK_MECHANISM mechanism_der =
                new CK_MECHANISM(CKM_GOSTR3410_PUBLIC_KEY_DERIVE);
            der_pub_key = pkcs11Wrapper.C_DeriveKey(
                recpSession, mechanism_der,
                unw_priv_key, pub_der_template);
            // Для проверки распакованного ключа
            // нужно что-нибудь им подписать
            // и проверить подпись открытым ключом.
            {
                CK_MECHANISM mechanism_sign =
                    new CK_MECHANISM(CKM_GOSTR3410);
                byte[] pData = {
                    (byte)0xDE, (byte)0xAD, (byte)0xBE, (byte)0xEF,
                    (byte)0xC0, (byte)0xC0, (byte)0xCA, (byte)0xFE,
                    (byte)0xDE, (byte)0xAD, (byte)0xBE, (byte)0xEF,
                    (byte)0xC0, (byte)0xC0, (byte)0xCA, (byte)0xFE,
                    (byte)0xDE, (byte)0xAD, (byte)0xBE, (byte)0xEF,
                    (byte)0xC0, (byte)0xC0, (byte)0xCA, (byte)0xFE,
                    (byte)0xDE, (byte)0xAD, (byte)0xBE, (byte)0xEF,
                    (byte)0xC0, (byte)0xC0, (byte)0xCA, (byte)0xFE
};

                // Sign with unwrapped private key
                pkcs11Wrapper.C_SignInit(recpSession,
                    mechanism_sign, unw_priv_key);
                byte[] signature =
                    pkcs11Wrapper.C_Sign(recpSession, pData);

                // Now try to verify with source public key
                pkcs11Wrapper.C_VerifyInit(sendSession,
                    mechanism_sign, pub_key);
                pkcs11Wrapper.C_Verify(sendSession, pData, signature);

                // Now try to verify with derived public key
```

```
                pkcs11Wrapper.C_VerifyInit(
                    recpSession, mechanism_sign, der_pub_key);
                pkcs11Wrapper.C_Verify(recpSession, pData, signature);
            }
            pkcs11Wrapper.C_DestroyObject(recpSession, der_pub_key);
            pkcs11Wrapper.C_DestroyObject(recpSession, unw_priv_key);
            pkcs11Wrapper.C_DestroyObject(sendSession, pub_key);
            pkcs11Wrapper.C_DestroyObject(sendSession, priv_key);

            System.out.println("SUCCESS");
}
catch(Exception e){
            System.out.println(e.getMessage());
}
    }
}
```

3.12. Генерация ключа по паролю

```
package pkcs11Test.pkcs11PBA;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import java.util.Arrays;
import pkcs11Test.utils.*;

public class PKCS11PBA {
    private final static byte[] STR_CRYPTO_PRO_HASH1 = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01
    };
    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };

    public static void main(String[] args) {
        System.out.println(
            "CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC test");
try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
            // Данный метод был изменен в Java 1.6
```

```
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.err.println("No slots found.");
System.exit(0);
            }

            // Открываем сессию
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);
            // Предъявляем ПИН-код пользователя.
            pkcs11Wrapper.C_Login(session, CKU_USER, Pin);

            // Real PKCS#12 params and ethalon
            byte[] salt4 = {
                (byte)0xDF, (byte)0x7C, (byte)0x5C, (byte)0x10,
                (byte)0xA4, (byte)0xD5, (byte)0x22, (byte)0x62
            };
            long iter4 = 2048;
            // Use UTF-8 password here
            // Converting from UTF-8 to UTF-16 in soft token
            char[] password4 = {'4', '4', '4', '4'};
            byte[] et14 = {
                (byte)0x93, (byte)0x88, (byte)0x91, (byte)0x11,
                (byte)0x20, (byte)0x43, (byte)0xC4, (byte)0xD1,
                (byte)0xCA, (byte)0x23, (byte)0x82, (byte)0xEF,
                (byte)0x86, (byte)0x4A, (byte)0x67, (byte)0x31,
                (byte)0xBD, (byte)0x29, (byte)0xEF, (byte)0x82,
                (byte)0x94, (byte)0xDD, (byte)0x23, (byte)0x50,
                (byte)0x63, (byte)0x0B, (byte)0xCB, (byte)0x1E,
                (byte)0x5A, (byte)0xC9, (byte)0x06, (byte)0xC3
```

```
            };

            CK_ATTRIBUTE[] attr_key_type = {
                new CK_ATTRIBUTE(CKA_KEY_TYPE)
            };
            CK_ATTRIBUTE[] attr_value = {
                new CK_ATTRIBUTE(CKA_VALUE)
            };
            CK_ATTRIBUTE[] key_template = {
                new CK_ATTRIBUTE(CKA_SENSITIVE, false ),
                new CK_ATTRIBUTE(CKA_EXTRACTABLE, true )
            };
            CK_GOSTR3411_PBE_PARAMS params =
                new CK_GOSTR3411_PBE_PARAMS(
                    STR_CRYPTO_PRO_HASH1, password4, salt4, iter4 );
            CK_MECHANISM mechanism = new CK_MECHANISM(
                CKM_PBA_GOSTR3411_WITH_GOSTR3411_HMAC, params);
            long key = pkcs11Wrapper.C_GenerateKey(
                session, mechanism, key_template);

            pkcs11Wrapper.C_GetAttributeValue(session, key, attr_key_type);
            long KeyType = attr_key_type[0].getLong();
            if (KeyType != CKK_GENERIC_SECRET) {
                System.err.println("Key type 0x"
                    + Functions.toHexString(KeyType)
                    + " != CKK_GENERIC_SECRET");
                System.exit(-1);
            }
            pkcs11Wrapper.C_GetAttributeValue(session, key, attr_value);
            byte[] KeyValue = attr_value[0].getByteArray();
            utils.hexDump("Generated key:", KeyValue);

            pkcs11Wrapper.C_DestroyObject(session, key);
            if (!Arrays.equals(KeyValue, et14)) {
                System.err.println("Generated key differs from ethalon");
                System.exit(-2);
            }
            System.out.println("Generated key equals to ethalon");
            System.out.println("SUCCESS");
        }
catch(Exception e){
            System.out.println(e.getMessage());
}
    }
```

```
}
```

3.13. Вычисление TLS PRF

```
package pkcs11Test.pkcs11TLS;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import java.util.Arrays;
import pkcs11Test.utils.*;

public class PKCS11TLSPRF {
    private final static byte[] STR_CRYPTO_PRO_HASH1 = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01
    };
    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };

    public static void main(String[] args) {
System.out.println("CKM_TLS_GOST_PRF test");
try{
     PKCS11 pkcs11Wrapper;
     CK_C_INITIALIZE_ARGS ck_c_initialize_args =
     new CK_C_INITIALIZE_ARGS();
     // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
     // Для Java 1.5 используется другой вызов
            /*
     pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */

     // Получаем список слотов, в которые вставлены токены.
     long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

     // Если ни одного токена не вставлено -- выходим.
     if(slotList.length == 0) {
System.err.println("No slots found.");
```

```
System.exit(-1);
     }

     // Открываем сессию.
     long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);
     // Предъявляем ПИН-код пользователя.
     pkcs11Wrapper.C_Login(session, CKU_USER, Pin);

            long oclass = CKO_SECRET_KEY;
            long key_type = CKK_GENERIC_SECRET;

            // Using master secret (48 bytes) from LirJSSE test utility.
            byte[] keyval = {
(byte)0x76,(byte)0x22,(byte)0x41,(byte)0x3d,
                (byte)0x5a,(byte)0xb6,(byte)0x7f,(byte)0x5b,
(byte)0x86,(byte)0x2e,(byte)0x75,(byte)0x97,
                (byte)0xe8,(byte)0xf9,(byte)0x31,(byte)0xe4,
(byte)0x2f,(byte)0x13,(byte)0x13,(byte)0xd5,
                (byte)0x61,(byte)0xed,(byte)0xab,(byte)0xed,
(byte)0xa5,(byte)0x4b,(byte)0xd8,(byte)0x5d,
                (byte)0x60,(byte)0x0a,(byte)0xec,(byte)0x99,
(byte)0x45,(byte)0xfe,(byte)0x39,(byte)0xa1,
                (byte)0xb7,(byte)0x62,(byte)0x0d,(byte)0x66,
(byte)0x3c,(byte)0xdd,(byte)0xb3,(byte)0x71,
                (byte)0x90,(byte)0x0f,(byte)0xc3,(byte)0xdd
            };
            CK_ATTRIBUTE[] key_template = {
new CK_ATTRIBUTE(CKA_VALUE, keyval),
new CK_ATTRIBUTE(CKA_CLASS, oclass),
new CK_ATTRIBUTE(CKA_KEY_TYPE, key_type),
new CK_ATTRIBUTE(CKA_DERIVE, true)
            };

            // Using real TLS message label from LirJSSE test utility.
            byte[] label = "client finished".getBytes();

            // seed (64 bytes) = client_random (32 bytes)
            // + server_random (32 bytes)

            // Using concatenated client_random and
            // server_random data (32+32 bytes)
```

```
            // from LirJSSE test utility.
            byte[] seed = {
(byte)0x87,(byte)0x5a,(byte)0x38,(byte)0x94,
                (byte)0xa4,(byte)0xf9,(byte)0x3c,(byte)0x30,
(byte)0x65,(byte)0xfc,(byte)0x66,(byte)0xbe,
                (byte)0xf2,(byte)0xc0,(byte)0x09,(byte)0xb9,
(byte)0xc3,(byte)0x26,(byte)0x3e,(byte)0x16,
                (byte)0xc7,(byte)0x28,(byte)0x14,(byte)0x27,
(byte)0x98,(byte)0x26,(byte)0xe4,(byte)0xd5,
                (byte)0x30,(byte)0x7f,(byte)0x9a,(byte)0x2d,
(byte)0x87,(byte)0x0e,(byte)0x1d,(byte)0x34,
                (byte)0xaf,(byte)0x28,(byte)0x1c,(byte)0x60,
(byte)0xae,(byte)0x8d,(byte)0x26,(byte)0xe4,
                (byte)0xd5,(byte)0x30,(byte)0x7f,(byte)0x9a,
(byte)0x2d,(byte)0x87,(byte)0x0e,(byte)0x1d,
                (byte)0x34,(byte)0xaf,(byte)0x28,(byte)0x1c,
(byte)0x60,(byte)0xae,(byte)0x8d,(byte)0x21,
                (byte)0xee,(byte)0x7d,(byte)0x52,(byte)0x44
            };
            // Using resulting TLS PRF value (12 bytes)
            // from LirJSSE test utility.
            byte[] et_gost = {
(byte)0x2a,(byte)0x6b,(byte)0x4c,(byte)0x14,
                (byte)0x61,(byte)0xb1,(byte)0x39,(byte)0x2d,
(byte)0x18,(byte)0xdf,(byte)0x8a,(byte)0x8d
            };
            byte[] value = new byte[12];

            CK_TLS_GOST_PRF_PARAMS params_gost =
                new CK_TLS_GOST_PRF_PARAMS(
                    new CK_TLS_PRF_PARAMS(seed, label, value),
                    STR_CRYPTO_PRO_HASH1
                );

            CK_MECHANISM mechanism =
                new CK_MECHANISM(CKM_TLS_GOST_PRF, params_gost);

            long keyh = pkcs11Wrapper.C_CreateObject(
             session, key_template);

            long dummyh = pkcs11Wrapper.C_DeriveKey(
                session, mechanism, keyh, null);

            utils.hexDump("GOST TLS PRF:", value);
```

```
            if(!Arrays.equals(value, et_gost)) {
                System.err.println(
                    "GOST TLS PRF value differs from ethalon");
                System.exit(-2);
            }
            System.out.println("GOST TLS PRF value equals to ethalon");

            pkcs11Wrapper.C_DestroyObject(session, keyh);

            pkcs11Wrapper.C_Logout(session);
            pkcs11Wrapper.C_CloseSession(session);
            System.out.println("SUCCESS");
}
catch(Exception e){
            System.out.println(e.getMessage());
}
    }
}
```

3.14. Генерация мастер-ключа TLS

```
package pkcs11Test.pkcs11TLS;

import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import java.util.Arrays;
import pkcs11Test.utils.*;

public class PKCS11TLSMASTER {
    private final static byte[] STR_CRYPTO_PRO_HASH1 = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01
    };
    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };

    public static void main(String[] args) {
System.out.println("CKM_TLS_GOST_MASTER_KEY_DERIVE test");
try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
```

```
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */

            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.err.println("No slots found.");
System.exit(-1);
            }

            // Открываем сессию.
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);
            // Предъявляем ПИН-код пользователя.
            pkcs11Wrapper.C_Login(session, CKU_USER, Pin);
            byte[] premaster = {
(byte)0xdc,(byte)0x2f,(byte)0x75,(byte)0x53,
                (byte)0x8a,(byte)0x02,(byte)0xc7,(byte)0x4d,
(byte)0x67,(byte)0x32,(byte)0x58,(byte)0xf5,
                (byte)0xf5,(byte)0x27,(byte)0xc3,(byte)0xc9,
(byte)0x27,(byte)0xd6,(byte)0x65,(byte)0x8f,
                (byte)0x72,(byte)0x38,(byte)0x7d,(byte)0x4b,
(byte)0x98,(byte)0x06,(byte)0xe0,(byte)0x91,
                (byte)0x1b,(byte)0xa5,(byte)0x8a,(byte)0x30
            };
            byte[] client_random = {
(byte)0x87,(byte)0x5a,(byte)0x38,(byte)0x94,
                (byte)0xa4,(byte)0xf9,(byte)0x3c,(byte)0x30,
(byte)0x65,(byte)0xfc,(byte)0x66,(byte)0xbe,
                (byte)0xf2,(byte)0xc0,(byte)0x09,(byte)0xb9,
(byte)0xc3,(byte)0x26,(byte)0x3e,(byte)0x16,
                (byte)0xc7,(byte)0x28,(byte)0x14,(byte)0x27,
```

```
(byte)0x98,(byte)0x26,(byte)0xe4,(byte)0xd5,
                (byte)0x30,(byte)0x7f,(byte)0x9a,(byte)0x2d
            };
            byte[] server_random = {
(byte)0x87,(byte)0x0e,(byte)0x1d,(byte)0x34,
                (byte)0xaf,(byte)0x28,(byte)0x1c,(byte)0x60,
(byte)0xae,(byte)0x8d,(byte)0x26,(byte)0xe4,
                (byte)0xd5,(byte)0x30,(byte)0x7f,(byte)0x9a,
(byte)0x2d,(byte)0x87,(byte)0x0e,(byte)0x1d,
                (byte)0x34,(byte)0xaf,(byte)0x28,(byte)0x1c,
(byte)0x60,(byte)0xae,(byte)0x8d,(byte)0x21,
                (byte)0xee,(byte)0x7d,(byte)0x52,(byte)0x44
            };
            byte[] master = {
(byte)0x92,(byte)0xf3,(byte)0x91,(byte)0xbc,
                (byte)0xe7,(byte)0x04,(byte)0xe3,(byte)0xbd,
(byte)0x1c,(byte)0x57,(byte)0x6d,(byte)0x8c,
                (byte)0x55,(byte)0x7b,(byte)0x54,(byte)0x2a,
(byte)0x82,(byte)0x52,(byte)0x75,(byte)0xfc,
                (byte)0x79,(byte)0x64,(byte)0xcc,(byte)0xcb,
(byte)0xb3,(byte)0x21,(byte)0x41,(byte)0xce,
                (byte)0x50,(byte)0x17,(byte)0x23,(byte)0x74,
(byte)0xdd,(byte)0xb9,(byte)0xa1,(byte)0xb3,
                (byte)0x5b,(byte)0xc2,(byte)0x80,(byte)0xab,
(byte)0xd5,(byte)0xec,(byte)0x62,(byte)0x70,
                (byte)0x63,(byte)0x6b,(byte)0xcb,(byte)0xb1
            };

            CK_ATTRIBUTE[] key_gen_template = {
new CK_ATTRIBUTE(CKA_VALUE, premaster),
new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GENERIC_SECRET),
new CK_ATTRIBUTE(CKA_SENSITIVE, false),
new CK_ATTRIBUTE(CKA_EXTRACTABLE, true),
new CK_ATTRIBUTE(CKA_DERIVE, true),
new CK_ATTRIBUTE(CKA_SIGN, true),
new CK_ATTRIBUTE(CKA_VERIFY, true)
            };
            CK_ATTRIBUTE[] key_derive_template = {
new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GENERIC_SECRET),
new CK_ATTRIBUTE(CKA_SENSITIVE, false),
new CK_ATTRIBUTE(CKA_EXTRACTABLE, true),
new CK_ATTRIBUTE(CKA_DERIVE, true),
```

```
new CK_ATTRIBUTE(CKA_SIGN, true),
new CK_ATTRIBUTE(CKA_VERIFY, true)
            };

            CK_TLS_GOST_MASTER_KEY_DERIVE_PARAMS params_gost =
                new CK_TLS_GOST_MASTER_KEY_DERIVE_PARAMS(
                    new CK_SSL3_RANDOM_DATA(client_random, server_random),
                    STR_CRYPTO_PRO_HASH1);

            long keyh = pkcs11Wrapper.C_CreateObject(
                session, key_gen_template);

            CK_MECHANISM mechanism =
                new CK_MECHANISM(CKM_TLS_GOST_MASTER_KEY_DERIVE,
                    params_gost);
            long mkeyh = pkcs11Wrapper.C_DeriveKey(
                    session, mechanism, keyh, key_derive_template);
            CK_ATTRIBUTE[] attr = {
                new CK_ATTRIBUTE(CKA_VALUE)
            };
            pkcs11Wrapper.C_GetAttributeValue(session, mkeyh, attr);
            byte[] value = attr[0].getByteArray();

            pkcs11Wrapper.C_DestroyObject(session, mkeyh);
            pkcs11Wrapper.C_DestroyObject(session, keyh);

            if(!Arrays.equals(value, master)) {
                System.err.println("Master value differs from ethalon");
                System.exit(-2);
            }
            System.out.println("Master key equals to ethalon");
            System.out.println("SUCCESS");
}
catch(Exception e){
            System.out.println(e.getMessage());
}
    }
}
```

3.15. Генерация ключей и имитовставок TLS

```
package pkcs11Test.pkcs11TLS;
```

```
import ru.lissi.security.pkcs11.wrapper.*;
import static ru.lissi.security.pkcs11.wrapper.PKCS11Constants.*;
import java.util.Arrays;

public class PKCS11TLSKEYMAC {
    private final static byte[] STR_CRYPTO_PRO_HASH1 = {
        (byte)0x06, (byte)0x07, (byte)0x2A, (byte)0x85,
        (byte)0x03, (byte)0x02, (byte)0x02, (byte)0x1e, (byte)0x01
    };
    private final static char[] Pin = {
        '0', '1', '2', '3', '4', '5', '6', '7'
    };

    public static void main(String[] args) {
        System.out.println("CKM_TLS_GOST_KEY_AND_MAC_DERIVE test");
try{
            PKCS11 pkcs11Wrapper;
            CK_C_INITIALIZE_ARGS ck_c_initialize_args =
                new CK_C_INITIALIZE_ARGS();
            // Данный метод был изменен в Java 1.6
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                "C_GetFunctionList", ck_c_initialize_args, false);
            // Для Java 1.5 используется другой вызов
            /*
            pkcs11Wrapper = PKCS11.getInstance(
                "lccryptoki_fs",
                ck_c_initialize_args, false);
            */
            // Получаем список слотов, в которые вставлены токены.
            long[] slotList = pkcs11Wrapper.C_GetSlotList(true);

            // Если ни одного токена не вставлено -- выходим.
            if(slotList.length == 0) {
System.err.println("No slots found.");
System.exit(-1);
            }

            // Открываем сессию отправителя.
            long session = pkcs11Wrapper.C_OpenSession(
                slotList[0],
                CKF_SERIAL_SESSION | CKF_RW_SESSION,
                null, null);
            pkcs11Wrapper.C_Login(session, CKU_USER, Pin);
```

```
            byte[] client_random = {
(byte)0x48,(byte)0xdc,(byte)0xdb,(byte)0x79,
                (byte)0xfb,(byte)0x11,(byte)0x16,(byte)0xaf,
(byte)0xed,(byte)0x6a,(byte)0x72,(byte)0xda,
                (byte)0xbf,(byte)0x7a,(byte)0xb2,(byte)0x76,
(byte)0x8c,(byte)0x35,(byte)0xa7,(byte)0x54,
                (byte)0x52,(byte)0xda,(byte)0xa1,(byte)0x00,
(byte)0x47,(byte)0x83,(byte)0x29,(byte)0x72,
                (byte)0xa9,(byte)0x8d,(byte)0xd9,(byte)0x78
            };
            byte[] server_random = {
(byte)0x48,(byte)0xdc,(byte)0xda,(byte)0xaa,
                (byte)0x83,(byte)0xeb,(byte)0x5c,(byte)0x08,
(byte)0x51,(byte)0x20,(byte)0xc6,(byte)0x8e,
                (byte)0x29,(byte)0xb4,(byte)0x30,(byte)0xff,
(byte)0xc5,(byte)0x9e,(byte)0x4f,(byte)0x1c,
                (byte)0xa9,(byte)0x75,(byte)0xa7,(byte)0x35,
(byte)0x62,(byte)0x89,(byte)0x29,(byte)0x17,
                (byte)0x28,(byte)0x70,(byte)0x56,(byte)0x06
            };
            byte[] master = {
(byte)0x68,(byte)0x0d,(byte)0x31,(byte)0x53,
                (byte)0x6a,(byte)0x20,(byte)0x68,(byte)0x50,
(byte)0xca,(byte)0x66,(byte)0x01,(byte)0x3f,
                (byte)0xb6,(byte)0xfa,(byte)0xe3,(byte)0x26,
(byte)0x51,(byte)0xb5,(byte)0xba,(byte)0x03,
                (byte)0x16,(byte)0xa3,(byte)0xdf,(byte)0xc1,
(byte)0x33,(byte)0x99,(byte)0xd0,(byte)0x61,
                (byte)0xda,(byte)0x74,(byte)0x12,(byte)0x4e,
(byte)0x96,(byte)0x6f,(byte)0x9a,(byte)0x1c,
                (byte)0x25,(byte)0x24,(byte)0x89,(byte)0x42,
(byte)0xa9,(byte)0xeb,(byte)0x0c,(byte)0x56,
                (byte)0xe1,(byte)0x04,(byte)0xbb,(byte)0x6a
            };
            byte[] ClientMacSecret = {
(byte)0x12,(byte)0x9d,(byte)0xf5,(byte)0xe1,
                (byte)0x00,(byte)0xfe,(byte)0x51,(byte)0xf5,
(byte)0xe9,(byte)0xf3,(byte)0xcf,(byte)0xff,
                (byte)0x4f,(byte)0x02,(byte)0xcc,(byte)0xc9,
(byte)0x4a,(byte)0xe8,(byte)0x32,(byte)0x11,
                (byte)0xbd,(byte)0x35,(byte)0x63,(byte)0xa8,
(byte)0xe2,(byte)0xa9,(byte)0x71,(byte)0xd2,
                (byte)0x61,(byte)0x24,(byte)0x7b,(byte)0x23
```

```
            };
            byte[] ServerMacSecret = {
(byte)0xc0,(byte)0xfc,(byte)0x04,(byte)0x6f,
                (byte)0xa2,(byte)0xc6,(byte)0x30,(byte)0x97,
(byte)0xa2,(byte)0x90,(byte)0x8a,(byte)0x47,
                (byte)0x56,(byte)0x26,(byte)0x9d,(byte)0xc9,
(byte)0xe9,(byte)0x87,(byte)0x71,(byte)0x41,
                (byte)0xc4,(byte)0x3d,(byte)0x81,(byte)0x81,
(byte)0xb0,(byte)0x30,(byte)0x8f,(byte)0xea,
                (byte)0xa3,(byte)0x86,(byte)0x4f,(byte)0xae
            };
            byte[] ClientKey = {
(byte)0x0e,(byte)0xfc,(byte)0x85,(byte)0x03,
                (byte)0x2b,(byte)0x40,(byte)0xbd,(byte)0x65,
(byte)0xce,(byte)0x39,(byte)0x70,(byte)0xb0,
                (byte)0xed,(byte)0xd2,(byte)0x48,(byte)0xdd,
(byte)0x6e,(byte)0xfc,(byte)0x97,(byte)0x29,
                (byte)0xb4,(byte)0xd5,(byte)0xd8,(byte)0x91,
(byte)0x42,(byte)0x7a,(byte)0xd0,(byte)0x55,
                (byte)0x9c,(byte)0x7c,(byte)0x6e,(byte)0x61
            };
            byte[] ServerKey = {
(byte)0x17,(byte)0x34,(byte)0xe4,(byte)0x2c,
                (byte)0x8e,(byte)0x9c,(byte)0x6f,(byte)0xda,
(byte)0x3c,(byte)0x30,(byte)0xbb,(byte)0xc8,
                (byte)0xfa,(byte)0xe7,(byte)0x8d,(byte)0x76,
(byte)0x11,(byte)0x0c,(byte)0xfc,(byte)0x5b,
                (byte)0xeb,(byte)0x00,(byte)0x92,(byte)0xb2,
(byte)0xed,(byte)0x36,(byte)0x3e,(byte)0x27,
                (byte)0x60,(byte)0x12,(byte)0x50,(byte)0xa5
            };
            byte[] IVClient = {
(byte)0x99,(byte)0x38,(byte)0xb5,(byte)0x71,
                (byte)0x78,(byte)0x47,(byte)0x92,(byte)0xba
            };
            byte[] IVServer = {
(byte)0xc4,(byte)0x43,(byte)0xb9,(byte)0xd3,
                (byte)0x24,(byte)0x75,(byte)0x09,(byte)0x79
            };
            CK_ATTRIBUTE[] key_gen_template = {
new CK_ATTRIBUTE(CKA_VALUE, master),
new CK_ATTRIBUTE(CKA_CLASS, CKO_SECRET_KEY),
new CK_ATTRIBUTE(CKA_DERIVE, true),
new CK_ATTRIBUTE(CKA_SIGN, true),
```

```
new CK_ATTRIBUTE(CKA_VERIFY, true),
new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GENERIC_SECRET),
new CK_ATTRIBUTE(CKA_SENSITIVE, false),
new CK_ATTRIBUTE(CKA_EXTRACTABLE, true)
            };
            CK_ATTRIBUTE[] key_derive_template = {
new CK_ATTRIBUTE(CKA_KEY_TYPE, CKK_GOST28147),
new CK_ATTRIBUTE(CKA_SENSITIVE, false),
new CK_ATTRIBUTE(CKA_EXTRACTABLE, true)
            };
            CK_TLS_GOST_KEY_MAT_PARAMS params_gost =
                new CK_TLS_GOST_KEY_MAT_PARAMS(
                    new CK_SSL3_KEY_MAT_PARAMS(
                        256, 256, 64, false,
                        new CK_SSL3_RANDOM_DATA(
                            client_random,
                            server_random
                        )
                    ),
                    STR_CRYPTO_PRO_HASH1
                );
            CK_MECHANISM mechanism =
                new CK_MECHANISM(
                    CKM_TLS_GOST_KEY_AND_MAC_DERIVE, params_gost);
            long keyh = pkcs11Wrapper.C_CreateObject(
                session, key_gen_template);

            long dummyh = pkcs11Wrapper.C_DeriveKey(
                session, mechanism, keyh, key_derive_template);
            CK_ATTRIBUTE[] attr = {
                new CK_ATTRIBUTE(CKA_VALUE)
            };
            pkcs11Wrapper.C_GetAttributeValue(session,
                params_gost.KeyMatParams.pReturnedKeyMaterial.
                    hClientMacSecret,
                attr);
            byte[] value = attr[0].getByteArray();
            if (!Arrays.equals(value, ClientMacSecret)) {
                System.err.println(
                    "ClientMacSecret differs from ethalon");
                System.exit(-2);
            }
            System.out.println("ClientMacSecret equals to ethalon");
```

```
pkcs11Wrapper.C_GetAttributeValue(session,
    params_gost.KeyMatParams.pReturnedKeyMaterial.
        hServerMacSecret,
    attr);
value = attr[0].getByteArray();
if (!Arrays.equals(value, ServerMacSecret)) {
    System.err.println(
        "ServerMacSecret differs from ethalon");
    System.exit(-2);
}
System.out.println("ServerMacSecret equals to ethalon");

pkcs11Wrapper.C_GetAttributeValue(session,
    params_gost.KeyMatParams.pReturnedKeyMaterial.hClientKey,
    attr);
value = attr[0].getByteArray();
if (!Arrays.equals(value, ClientKey)) {
    System.err.println("ClientKey differs from ethalon");
    System.exit(-2);
}
System.out.println("ClientKey equals to ethalon");

pkcs11Wrapper.C_GetAttributeValue(session,
    params_gost.KeyMatParams.pReturnedKeyMaterial.hServerKey,
    attr);
value = attr[0].getByteArray();
if (!Arrays.equals(value, ServerKey)) {
    System.err.println("ServerKey differs from ethalon");
    System.exit(-2);
}
System.out.println("ServerKey equals to ethalon");

if (!Arrays.equals(
    params_gost.KeyMatParams.pReturnedKeyMaterial.pIVClient,
    IVClient)) {
    System.err.println("IVClient differs from ethalon");
    System.exit(-2);
}
System.out.println("IVClient equals to ethalon");

if (!Arrays.equals(
    params_gost.KeyMatParams.pReturnedKeyMaterial.pIVServer,
    IVServer)) {
    System.err.println("IVServer differs from ethalon");
```

```
                System.exit(-2);
            }
            System.out.println("IVServer equals to ethalon");

            pkcs11Wrapper.C_DestroyObject(session,
                params_gost.KeyMatParams.pReturnedKeyMaterial.
                    hClientMacSecret);
            pkcs11Wrapper.C_DestroyObject(session,
                params_gost.KeyMatParams.pReturnedKeyMaterial.
                    hServerMacSecret);
            pkcs11Wrapper.C_DestroyObject(session,
                params_gost.KeyMatParams.pReturnedKeyMaterial.
                    hClientKey);
            pkcs11Wrapper.C_DestroyObject(session,
                params_gost.KeyMatParams.pReturnedKeyMaterial.
                    hServerKey);

            pkcs11Wrapper.C_DestroyObject(session, keyh);

            System.out.println("SUCCESS");
}
catch(Exception e){
            System.out.println(e.getMessage());
}
    }
}
```

A. Механизмы враппера LirPKCS11

В следующей таблице перечислены механизмы, поддерживаемые враппером LirPKCS11.

Механизмы
CKM_GOSTR3410_KEY_PAIR_GEN
CKM_GOSTR3410
CKM_GOSTR3410_WITH_GOSTR3411
CKM_GOSTR3410_DERIVE
CKM_GOSTR3410_KEY_WRAP
CKM_GOST28147_KEY_GEN
CKM_GOST28147
CKM_GOST28147_ECB
CKM_GOST28147_CNT
CKM_GOST28147_MAC
CKM_GOST28147_KEY_WRAP
CKM_GOST28147_PKCS8_KEY_WRAP
CKM_GOSTR3410_PUBLIC_KEY_DERIVE
CKM_GOSTR4311
CKM_GOSTR4311_HMAC
CKM_PBA_GOSTR4311_WITH_GOSTR3411_HMAC
CKM_PKCS5_PBKD2
CKM_TLS_GOST_PRF
CKM_TLS_GOST_PRE_MASTER_KEY_GEN
CKM_TLS_GOST_MASTER_KEY_DERIVE
CKM_TLS_GOST_KEY_AND_MAC_DERIVE
CKM_MD2_RSA_PKCS, CKM_RSA_PKCS, CKM_RSA_X_509
CKM_MD5_RSA_PKCS, CKM_RSA_PKCS, CKM_RSA_X_509
CKM_SHA1_RSA_PKCS, CKM_RSA_PKCS, CKM_RSA_X_509
CKM_SHA256_RSA_PKCS, CKM_RSA_PKCS, CKM_RSA_X_509
CKM_SHA384_RSA_PKCS, CKM_RSA_PKCS, CKM_RSA_X_509
CKM_SHA512_RSA_PKCS, CKM_RSA_PKCS, CKM_RSA_X_509
CKM_DSA_SHA1, CKM_DSA
CKM_DSA
CKM_ECDSA_SHA1, CKM_ECDSA

CKM_ECDSA
CKM_ECDSA
CKM_ECDSA
CKM_ECDSA
CKM_RSA_PKCS
CKM_RC4
CKM_DES_CBC
CKM_DES3_CBC
CKM_AES_CBC
CKM_BLOWFISH_CBC
CKM_ECDH1_DERIVE
CKM_DH_PKCS_DERIVE
CKM_RSA_PKCS_KEY_PAIR_GEN
CKM_DSA_KEY_PAIR_GEN
CKM_EC_KEY_PAIR_GEN
CKM_DH_PKCS_KEY_PAIR_GEN
CKM_RC4_KEY_GEN
CKM_DES_KEY_GEN
CKM_DES3_KEY_GEN
CKM_AES_KEY_GEN
CKM_BLOWFISH_KEY_GEN
CKM_MD5_HMAC
CKM_SHA_1_HMAC
CKM_SHA256_HMAC
CKM_SHA384_HMAC
CKM_SHA512_HMAC
CKM_MD2
CKM_MD5
CKM_SHA_1
CKM_SHA256
CKM_SHA384
CKM_SHA512
Любой поддерживаемый механизм RSA
Любой поддерживаемый механизм DSA
Любой поддерживаемый механизм EC
Любой поддерживаемый механизм Диффи-Хеллмана
CKM_RC4
CKM_DES_CBC
CKM_DES3_CBC
CKM_AES_CBC
CKM_BLOWFISH_CBC