УТВЕРЖДАЮ

Директор
ООО «ЛИССИ-Софт»

А.Степанов
« » 2013 г.

Средство Криптографической Защиты Информации «LCJCE»

Руководство разработчика

RU.ЛСАФ.00022-01 90 01

Листов 43

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Данное руководство предназначено для разработчиков прикладного ПО с использованием СКЗИ «LCJCE» и содержит описание криптопровайдера `LCJCE`.

2. ОБЩИЕ СВЕДЕНИЯ

Имя криптопровайдера	`LCJCE`
Имя библиотеки классов криптопровайдера	`lcjce.jar`
Мастер класс (*`masterClassName`*)	`ru.lc.security.provider.LCJCE`

3. ПОДДЕРЖИВАЕМЫЕ СТАНДАРТЫ

Криптопровайдер реализует российские алгоритмы криптографических преобразований в соответствии со следующими стандартами:

ГОСТ 28147-89	Алгоритм криптографического преобразования (шифрование и имитовставка)
ГОСТ Р 34.10-2001	Процессы формирования и проверки электронной цифровой подписи (ЭЦП)
ГОСТ Р 34.11-94	Функция хэширования

Криптопровайдер также поддерживает другие вспомогательные функции и алгоритмы, описанные ниже.

4. АРХИТЕКТУРА КРИПТОПРОВАЙДЕРА

Криптопровайдер построен в соответствии с требованиями Java Cryptography Architecture (JCA).

Java Cryptography Architecture (JCA) обеспечивает:

- независимость и совместимость реализации;
- независимость и расширяемость алгоритмов.

Независимость реализации и независимость алгоритмов в JCA взаимосвязаны и дополняют друг друга. Независимость алгоритмов достигается путем определения типов криптографических служб – «*engines*», и определения классов, которые обеспечивают функционал данных служб. Эти классы в JCA называются *engine classes - механизмы*. Примерами *механизмов* являются *MessageDigest*, *Signature*, *KeyFactory*, *KeyPairGenerator*.

Независимость реализации достигается использованием архитектуры на основе провайдеров криптографических служб. Провайдером является пакет или набор пакетов, реализующих одну или несколько криптографических служб. Приложения могут просто вызывать определенный тип объекта (например, объект Signature) реализованный одним из установленных провайдеров. Провайдеры могут обновляться независимо от приложений, их использующих.

Совместимость реализации означает, что разные провайдеры, реализующие одинаковые алгоритмы, могут использоваться разными приложениями и результаты их работы будут совместимыми.

Расширяемость алгоритмов означает возможность добавления поддержки провайдерами новых алгоритмов без переделки имеющихся приложений.

Криптографические службы определены абстрактным образом без реализации механизмов. Криптографическая служба всегда связана с определенным алгоритмом или типом и, либо выполняет криптографическую операцию (подпись, хэш), либо вырабатывает криптоматериал (ключи, параметры), либо создает объекты, в которых хранятся ключи защищенным образом.

Следующие *механизмы* реализованы в криптопровайдере LCJCE:

```
MessageDigest
Signature
KeyPairGenerator
KeyAgreement
```

```
KeyFactory
KeyStore
AlgorithmParameters
SecureRandom
Cipher
MAC
KeyGenerator
```

Классы *механизмов* предоставляют интерфейс к функционалу криптографической службы определенного типа независимо от конкретного алгоритма. Они определяют интерфейс, который позволяет приложениям использовать криптографические службы.

Для использования криптопровайдера необходимо вызвать метод `getInstance` соответствующего *механизма* с указанием имени алгоритма и опционально можно указать имя провайдера. Если провайдер не указывается, то выбирается наиболее подходящий провайдер для данного имени алгоритма, если он есть. Например,

```
Signature sign = Signature.getInstance("GOST_DS", "LCJCE");
```

Если провайдер не установлен возбуждается исключение `NoSuchProviderException`. Приложения могут получить список установленных и зарегистрированных в системе провайдеров с помощью вызова метода `getProviders` класса `Security`.

4.1. Идентификаторы объектов (OID)

Идентификаторы объектов используются для определения криптоалгоритмов и их параметров. В некоторых случаях JCA выбирает реализацию криптоалгоритма на основе идентификатора алгоритма полученного, например, из сертификата. Так, для проверки подписи сертификата X.509 на основе идентификатора алгоритма подписи из установленных криптопровайдеров выбирается и инициализируется соответствующий алгоритм проверки подписи.

Для того, чтобы JCA мог определить поддерживаемые провайдером алгоритмы, в нем должны быть определены отображения OID в соответствующие алгоритмы и наоборот.

В `LCJCE` определены следующие OIDы и соответствующие им алгоритмы:

Таблица. 1.

№	Наименование алгоритма, функции	Название *механизма* по JCA (JCA *engine class*)	Допустимые идентификаторы алгоритмов для методов `getInstance`	OID
1.	ЭЦП ГОСТ Р 34.10-2001	Signature	GOST_DS, GOST, GOST_2001, GOST_34_10_01.	1.2.643.2.2.3
2.	Выработка ключей ЭЦП ГОСТ Р 34.10-2001	KeyPairGenerator	GOST_DS, GDS_2001, GOST.	1.2.643.2.2.19
3.	Хэш ГОСТ Р 34.11-94	MessageDigest	HASH GOST GOST_HASH HASH_34_11_94	1.2.643.2.2.9
4.	Фабрика ключей	KeyFactory	GOST_DS GOST	1.2.643.2.2.19
5.	ГСЧ	SecureRandom	GOST GOST_RNG	1.2.643.2.18.2.1
6.	Хранилище ключей	KeyStore	LKS PKCS12	
7.	Шифрование ГОСТ 28147-89	Cipher	GOST GOST28147 CIPHER	1.2.643.2.2.21
8.	Имитовставка ГОСТ 28147-89	Mac	GOST	
9.	Выработка ключа шифрования ГОСТ 28147-89	KeyGenerator	GOST GOST28147	
10.	Параметры ГОСТ Р 34.10-2001	AlgorithmParameters	GOST_DS	1.2.643.2.2.19
11.	Параметры ГОСТ 28147-89	AlgorithmParameters	GOST28147 CIPHER	

5. ИНСТАЛЛЯЦИЯ ПРОВАЙДЕРА

Для инсталляции криптопровайдера необходимо установить пакет (библиотеку) классов провайдера и сконфигурировать его.

5.1. Установка библиотеки классов криптопровайдера

Для установки LCJCE необходимо сделать доступной для приложений библиотеку классов криптопровайдера: `lcjce.jar`. Для этого есть два пути:

- Установить `lcjce.jar` в любой из каталогов, определенных в системной переменной CLASSPATH или в параметре командной строки `classpath`.
- Установить lcjce.jar в следующий каталог:

```
<java-home>\lib\ext                    [Windows]
<java-home>/lib/ext                    [Unix]
```

Здесь `<java-home>` означает каталог, в котором установлена JRE.

Заметим, что при обращении к некоторым алгоритмам криптопровайдера через интерфейсы JCA производится его аутентификация. Поэтому соответствующий JAR-файл подписан специальным сертификатом разработчика - ООО "ЛИССИ-Крипто".

5.2 Конфигурация криптопровайдера

Необходимо добавить провайдер в список используемых (зарегистрировать). Для того, чтобы зарегистрировать провайдер статически (постоянно), необходимо отредактировать файл свойств безопасности (*security properties file*):

```
<java-home>\lib\security\java.security        [Win32]
<java-home>/lib/security/java.security      [Unix]
```

Здесь `<java-home>` означает каталог, в котором установлена JRE.

Каждый зарегистрированный провайдер должен быть описан в данном файле строкой вида:

```
    security.provider.n=masterClassName
```

Таким образом определяется провайдер и его приоритет. Приоритет провайдера определяется его порядковым номером n. Порядковые номера провайдеров начинаются с единицы, означающей наибольший приоритет.

`masterClassName` – есть имя мастер-класса провайдера.

Криптопровайдер `lcjce.jar` должен быть добавлен к списку провайдеров с любым доступным последовательным номером, например:

`security.provider.7=ru.lc.security.provider.LCJCE.`

Провайдер может также регистрироваться динамически. Для этого могут использоваться методы `addProvider()` или `insertProviderAt()` класса `Security`. Данный тип регистрации не является постоянным и может быть сделан только доверенным приложением (См. раздел `Security class` в документации Sun - *Java Cryptography Architecture API Specification and Reference*).

5.3. Методы класса Provider

Каждый провайдер имеет чувствительное к регистру имя, версию, и строку описания. Для получения описания провайдера используются методы:

```
public String getName()
public double getVersion()
public String getInfo()
```

5.4. Управление провайдерами

В следующих таблицах приведены методы класса *`Security`* для управления провайдерами.

Таблица. 2

Опрос провайдеров	
Метод	**Описание**
`static Provider[] getProviders()`	Возвращает массив, содержащий все установленные провайдеры. Порядок провайдеров в массиве соответствует их порядку при просмотре во время поиска подходящего провайдера.
`static Provider getProvider (String providerName)`	Возвращает *`Provider`* с именем *`providerName`*. Если *`Provider`* не найден возвращается *`null`*.

Таблица. 3

Добавление провайдеров	
Метод	**Описание**
`static int addProvider(Provider provider)`	Добавляет *Provider* в конец списка установленных провайдеров. Возвращает позицию в списке, если провайдер добавлен, или -1 если *Provider* не добавлен из-за того, что он уже установлен.
`static int insertProviderAt (Provider provider, int position)`	Добавляет провайдера *Provider* в определенную позицию. Все, стоящие за указанной позицией провайдеры сдвигаются на одну позицию. Возвращает позицию в списке, если провайдер добавлен, или -1 если *Provider* не добавлен из-за того, что он уже установлен.

Удаление провайдеров	
Метод	**Описание**
`static void removeProvider(String name)`	Удаляет *Provider* с именем *name*. Все провайдеры смещаются на одну позицию к началу списка.

Замечание: Если надо поменять позицию провайдера в списке, то для этого его надо сначала удалить, а затем вставить в нужную позицию.

5.5. Примеры программ

В составе LCJCE SDK имеются примеры программ, демонстрирующие использование различных алгоритмов провайдера, а также исходные данные для выполнения этих программ. Настоятельно рекомендуется ознакомиться с исходными текстами примеров перед разработкой собственных прикладных программ.

6. КЛАССЫ И ИНТЕРФЕЙСЫ СПЕЦИФИКАЦИЙ ПАРАМЕТРОВ АЛГОРИТМОВ

Спецификация параметров алгоритма есть подробное представление набора параметров, используемых алгоритмом. Подробное представление означает, что можно получить и изменить каждый параметр отдельно. Доступ к параметрам осуществляется с помощью методов *get*, определенных в соответствующем классе спецификации. Например, класс `Gost_28147ParameterSpec` имеет метод `getTZ()` для получения таблицы замен.

Кроме подробного представления параметров существует общее представление, которое не предоставляет доступа к полям параметров. Можно только получить имя алгоритма связанного с параметрами (через метод *getAlgorithm*) и некоторое кодовое представление набора параметров (через метод *getEncoded*).

Общие интерфейсы и классы спецификации параметров алгоритмов JCE определены в пакете *java.security.spec*. Специфичные для алгоритмов, реализованных в провайдере LCJCE интерфейсы параметров алгоритмов определены в пакете *ru.lc.security.crypto.interfaces*, а классы в пакете *ru.lc. security.crypto.spec*.

Провайдер LCJCE содержит в пакете `ru.lc.security.crypto.interfaces`, входящем в `lcjce.jar`, интерфейсы и классы, необходимые для работы с параметрами реализуемых криптоалгоритмов:

```
Gost_28147Params
Gost_DSParams
Gost_DSKey
Gost_DSPublicKey
Gost_DSPrivateKey
```

На базе данных интерфейсов реализованы классы спецификаций ключей и параметров в пакете `ru.lc.security.crypto.spec`:

```
Gost_DSKeySpec
Gost_PrivKeySpec
Gost_PubKeySpec
DHParameterSpec
```

Эти спецификации используются в фабриках ключей (KeyFactory) или в алгоритме генерации ключей согласования (KeyAgreement). Фабрики ключей осуществляют двухстороннее преобразование между общим представлением ключа (типа *Key*) и его спецификацией. Ключ согласования генерируется на основе закрытого ключа отправителя и открытого ключа получателя или на основе закрытого ключа получателя и открытого ключа отправителя.

6.1. Интерфейс java.security.spec.AlgorithmParameterSpec

AlgorithmParameterSpec – это интерфейс для подробного представления параметров криптоалгоритмов. Этот интерфейс не содержит методов и констант. Основная его цель – дать общий тип для всех спецификаций параметров. Все спецификации параметров должны реализовывать данный интерфейс.

6.2. Класс Gost_28147ParameterSpec

Данный класс реализует интерфейсы *Gost_28147Params* и *AlgorithmParameterSpec* и определяет набор параметров алгоритма ГОСТ 28147-89. Параметрами алгоритма шифрования ГОСТ 28147-89 являются узлы замен (подстановок). Класс *Gost_28147ParameterSpec* имеет два конструктора:

```
public Gost_28147ParameterSpec(byte[] tz)
public Gost_28147ParameterSpec(byte[] tz, byte[] iv, boolean
meshing)
```

где параметр *tz[]* определяет массив узла замен длинной 64 байта, параметр *iv* определяет начальное значение вектора инициализации (синхропосылки 8 байт), а параметр *meshing* позволяет определить необходимость модификации ключа (key meshing) после обработки каждого килобайта данных. Модификация ключа осуществляется в соответствии с рекомендациями КриптоПро. Если *meshing* равен true, то модификация ключа будет осуществляться объектом *Cipher*, который был проинициализирован с использованием данного экземпляра класса *Gost_28147ParameterSpec* (См. также раздел Класс Cipher).

Если для создания объекта класса *Gost_28147ParameterSpec* используется конструктор только с параметром *tz*, то используется нулевой вектор инициализации и модификация ключа не применяется.

Класс *Gost_28147ParameterSpec* имеет методы:

`public byte[] getTZ()` – возвращает массив узла замен;

`public String getOID()` – устанавливает OID узла замен;

`public void setOID(String OID)` – возвращает OID узла замен.

`public byte[] getIV()` – возвращает начальное значение вектора инициализации (синхропосылки).

`public void setIV(byte[] iv)` – устанавливает начальное значение вектора инициализации (синхропосылки).

`public boolean getMeshing()` – возвращает значение параметра *meshing*.

`public void setMeshing(boolean meshing)` – устанавливает значение параметра *meshing*.

6.3. Класс Gost_DSParameterSpec

Данный класс реализует интерфейсы *Gost_DSParams* и *AlgorithmParameterSpec* и определяет набор параметров алгоритма ГОСТ Р 34.10-2001.

Параметры алгоритма ЭЦП ГОСТ Р 34.10-2001 включают:

`a` и `b` – параметры эллиптической кривой,

`p` – модуль группы точек эллиптической кривой,

`q` – порядок группы точек эллиптической кривой,

`x` и `y` – координаты базовой точки.

Конструктор класса имеет вид:

```
public Gost_DSParameterSpec(
    BigInteger p,
    BigInteger a,
    BigInteger b,
    BigInteger q,
    BigInteger x,
    BigInteger y,
    Gost_28147ParameterSpec cp)
```

Методами класса *Gost_DSParameterSpec* являются:

```
public BigInteger getA()
public BigInteger getB()
```

```
    public BigInteger getP()
    public BigInteger getQ()
    public BigInteger getX()
    public BigInteger getY()
    public Gost_28147Params getCipherPar()
    public String getOID()
    public String getHASH_OID()
    public void setHASH_OID(String hashOID)
    public void setOID(String dsOID)
```

6.4. Класс AlgorithmParameters

Класс *AlgorithmParameters* является *механизмом*, который дает общее представление криптопараметров.

Общее представление не предоставляет доступа к полям параметров. Существует возможность только получить имя алгоритма, соответствующего параметрам и установить тип кодирования набора параметров для экспорта. Существует метод *AlgorithmParameters getParameterSpec* для преобразования объекта *AlgorithmParameters* в подробную спецификацию.

Как и для всех других *механизмов*, объект *AlgorithmParameters* получается с помощью одного из методов *getInstance*.

```
static AlgorithmParameters getInstance(String algorithm)
static AlgorithmParameters getInstance(String algorithm, String
provider)
static AlgorithmParameters getInstance(String algorithm, Pro-
vider provider)
```

Строка *algorithm* должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для *AlgorithmParameters*.

После получения объекта его необходимо проинициализировать с помощью одного из методов *init*:

```
void init(AlgorithmParameterSpec paramSpec)
void init(byte[] params)
void init(byte[] params, String format)
```

В этих методах *init*, поле params содержит кодированные параметры, а поле *format* есть имя формата кодировки параметров. Формат кодировки параметров по умолчанию – ASN.1.

Для получения закодированных в ASN.1 параметров объекта *AlgorithmParameters* используется метод *getEncoded*:

```
byte[] getEncoded()
byte[] getEncoded(String format)
```

В реализации провайдера LCJCE в методах *init* и *getEncoded* параметр *format* игнорируется.

6.4.1. Преобразование объекта *AlgorithmParameters* в подробную спецификацию

Подробная спецификация параметров алгоритмов может быть получена из объекта *AlgorithmParameters* с помощью метода *getParameterSpec*:

```
AlgorithmParameterSpec getParameterSpec(Class paramSpec)
```

Поле *paramSpec* определяет класс спецификации, в которой параметры должны быть возвращены. Например, *Gost_DSParameterSpec.class* для того, чтобы параметры были возвращены в классе *Gost_DSParameterSpec*.

7. ИНТЕРФЕЙСЫ И КЛАССЫ КЛЮЧЕЙ И КЛЮЧЕВЫХ СПЕЦИФИКАЦИЙ

7.1. Интерфейс **Key**

Интерфейс *Key* – есть интерфейс верхнего уровня для всех ключей представленных в общем виде.

Для ключей в общем виде нет возможности доступа к ключевому материалу. Всего три метода определены в интерфейсе *Key*: *getAlgorithm*, *getFormat* и *getEncoded*.

```
String getAlgorithm()
byte[] getEncoded()
String getFormat()
```

Метод *getEncoded* необходим для экспорта ключа, например в форматах X.509 или PKCS #8.

7.2. Классы ключевых спецификаций

Ключевая спецификация – есть развернутое представление ключа, т.е. есть подробное представление составных частей ключевого материала. Например, если ключ хранится в каком-нибудь устройстве, то спецификация может содержать информацию, помогающую идентифицировать ключ в устройстве.

В отличии от развернутого, общее представление ключей, определенное в интерфейсе *Key*, не предоставляет прямого доступа к полям ключей. Интерфейс *Key* имеет только три метода: *getAlgorithm*, *getFormat*, и *getEncoded*. Метод *getEncoded* позволяет производить экспорт ключевого материала в различных форматах, например PKCS8, ASN.1 и др.

Подробное представление ключа позволяет получить доступ отдельно к каждому параметру ключевого материала с помощью методов *«get»*, определенных в соответствующих классах спецификаций.

Например, в классе *ru.lc.security.crypto.spec.Gost_DSParameterSpec* определены методы *getP*, *getA*, *getB*, *getQ*, *getX* и *getY* - для доступа к параметрам алгоритма ГОСТ Р 34.10-2001, *getCipherPar* – для доступа к параметрам ГОСТ 28147-89 и др. Класс *ru.lc.security.crypto.spec. Gost_DSKeySpec* включает поле типа *Gost_DSParameterSpec*. А на базе класса *Gost_DSKeySpec* построены классы *ru.lc.security.crypto.spec.Gost_PrivKeySpec* и *ru.lc.security.crypto.spec.Gost_PubKeySpec*. Класс *Gost_PrivKeySpec* имеет метод *getD*, а класс *Gost_PubKeySpec* – методы *getX* и *getY*, с помощью которых осуществляется доступ к данным ключей.

Класс *Gost_DSKeySpec* реализует интерфейс *java.security.spec.KeySpec*. Таким образом, классы *Gost_PrivKeySpec* и *Gost_PubKeySpec*, тоже косвенно реализуют интерфейс *java.security.spec.KeySpec*.

7.3. Класс KeyFactory

Класс *KeyFactory* является *механизмом*, который обеспечивает преобразование между общим представлением ключей (тип *Key*) и спецификациями ключей. Фабрики ключей обеспечивают преобразование в обе стороны.

Как и для всех других *механизмов*, объект *KeyFactory* получается с помощью одного из методов *getInstance*:

```
static KeyFactory getInstance(String algorithm)
static KeyFactory getInstance(String algorithm, String provider)
static KeyFactory getInstance(String algorithm, Provider pro-
vider)
```

Строка *algorithm* должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для класса *KeyFactory*.

7.3.1. Преобразование из спецификации ключа в объект Key

Если имеется спецификация публичного ключа, можно получить ключ в общем виде (объект *PublicKey*) с помощью метода *generatePublic*:

```
PublicKey generatePublic(KeySpec keySpec)
```

Аналогично из спецификации закрытого ключа, можно получить ключ в общем виде (объект *PrivateKey*) с помощью метода *generatePrivate*:

```
PrivateKey generatePrivate(KeySpec keySpec)
```

7.3.2. Преобразование объекта Key в спецификацию ключа

Данное преобразование осуществляется с помощью метода *getKeySpec*:

```
KeySpec getKeySpec(Key key, Class keySpec)
```

Здесь *keySpec* определяет спецификацию класса в котором должен быть получен ключевой материал объекта *key*.

Например, *Gost_PrivKeySpec.class*, указывает, что ключевой материал должен быть получен в виде класс спецификации ключа *Gost_PrivKeySpec*.

7.4. Класс KeyPair

Класс *KeyPair* просто объект для хранения пары закрытый-открытый ключ. Он имеет два публичных метода для получения этих ключей:

```
PrivateKey getPrivate()
PublicKey getPublic()
```

7.5. Класс KeyPairGenerator

Класс *KeyPairGenerator* является *механизмом*, который обеспечивает выработку пар ключей алгоритма ЭЦП.

Как и для всех других *механизмов*, объект *KeyPairGenerator* получается с помощью одного из методов *getInstance*.

```
static KeyPairGenerator getInstance(String algorithm)
static KeyPairGenerator getInstance(String algorithm, String provider)
static KeyPairGenerator getInstance(String algorithm, Provider provider)
```

Строка *algorithm* должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для класса *KeyPairGenerator*.

Функция выработки пар ключей ЭЦП, поддерживаемая провайдером LCJCE, вырабатывает два ключа: открытый как массив из 64 байт, который представляет координату точки эллиптической кривой в виде двух целых x и y, и закрытый как массив из 32 байт.

7.5.1. Инициализация *KeyPairGenerator*

Генератор ключей требует инициализации. Существуют несколько методов инициализации:

```
void initialize(int keysize, SecureRandom random)
void initialize(int keysize)
void initialize(AlgorithmParameterSpec params, SecureRandom random)
void initialize(AlgorithmParameterSpec params)
```

В криптопровайдере LCJCE параметр *keysize* всегда должен быть равен 256 (длине в битах закрытого ключа), т.к. размеры ключей строго определены ГОСТ.

Параметр *random* определяет объект генерации случайных чисел (ГСЧ). Если используется метод инициализации без параметра *random*, то используется системный генератор. Рекомендуется использовать для инициализации объекта *KeyPairGenerator* метод *initialize* с параметром *random*, который является объектом *SecureRandom* криптопровайдера LCJCE.

Параметр *params* позволяет задавать параметры криптоалгоритма ГОСТ Р 34.10-2001. Если используется метод инициализации без параметра *params*, то используются параметры по умолчанию (см.ниже).

7.5.2. Параметры ключей криптоалгоритма ГОСТ Р 34.10-2001, выбираемые по умолчанию

Для криптопровайдера LCJCE существует возможность задать параметры криптоалгоритма ГОСТ Р 34.10-2001, которые будут использоваться по умолчанию при генерации ключей, а также выработке и проверке ЭЦП. Для задания параметров используется файл конфигурации – *lissi_prov.cf*. В файле конфигурации задаются:

- узел замен для вычисления хэша;
- параметры эллиптических кривых для генерации ключей.

Наличие файла *lissi_prov.cf* не является обязательным. При его отсутствии, будут использоваться следующие криптопараметры:

- узел замен для вычисления хэш-функции в соответствии с OID: 1.2.643.2.2.9[1] ; в соответствии с OID: 1.2.643.2.2.35.1 для ЭЦП.
- Необходимо отметить, что при обработке документов в форматах X.509 и PKCS будут использоваться криптопараметры алгоритмов, которые определены с помощью OID в данных документах.

7.5.3. Формат файла lissi_prov.cf

Файл *lissi_prov.cf* является простым текстовым файлом. Строки данного файла имеют вид: <имя параметра> = <значение>. Именами параметров могут быть: ecc_par – параметры ЭЦП, hash_par – параметры хэша. Значение параметров зависит от типа параметра и допускает использование синонимов, например, для указания параметров ЭЦП с OID 1 2 643 2 2 35 0 можно использовать строки:

```
ecc_par = id-GostR3410-2001-TestParamSet
ecc_par = 1 2 643 2 2 35 0
ecc_par = 1.2.643.2.2.35.0
ecc_par = 0
ecc_par = TestParamSet
ecc_par = test
```

Рекомендуемый способ задания параметров – через OID, разделенный точками (третья строка в примере выше). Перечни параметров и синонимов опре-

[1] Используются OID, приведенные в документе «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» компании КриптоПро.

делены в таблице Таблица. 4 для параметров ЭЦП и таблице Таблица. 5 для узлов замен.

Таблица. 4

Номер набора	Параметр	Шестнадцатеричное значение параметра	Символьные идентификаторы набора параметров
1	a	7	"id-GostR3410-2001-TestParamSet", "1 2 643 2 2 35 0", "1.2.643.2.2.35.0", "0", "test".
	b	5FBFF498AA938CE739B8E022FBAFEF40563F6E6A3472FC2A514C0CE9DAE23B7E	
	p	8000000000000000000000000000000000000000000000000000000000000431	
	q	8000000000000000000000000000000150FE8A1892976154C59CFC193ACCF5B3	
	x	2	
	y	8E2A8A0E65147D4BD6316030E16D19C85C97F0A9CA267122B96ABBCEA7E8FC8	
2*	a	p – 3	"id-GostR3410-2001-CryptoPro-A-ParamSet", "1 2 643 2 2 35 1", "1.2.643.2.2.35.1", "A", "1", "id-GostR3410-2001-CryptoPro-XchA-ParamSet", "1.2.643.2.2.36.0", "1 2 643 2 2 36 0", "XchA".
	b	a6	
	p	ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffd97	
	q	ffffffffffffffffffffffffffffffff6c611070995ad10045841b09b761b893	
	x	1	
	y	8d91e471e0989cda27df505a453f2b7635294f2ddf23e3b122acc99c9e9f1e14	
3	a	p – 3	

* Жирным шрифтом выделены наборы параметров, используемые по умолчанию.

	b	3e1af419a269a5f866a7d3c25c3df80ae979259373ff2b182f49d4ce7e1bbc8b	"id-GostR3410-2001-CryptoPro-B-ParamSet", "1 2 643 2 2 35 2", "1.2.643.2.2.35.2", "B", "2".
	p	8000000000000000000000000000000000000000000000000000000000000c99	
	q	8000000000000000000000000000000015f700cfff1a624e5e497161bcc8a198f	
	x	1	
	y	3fa8124359f96680b83d1c3eb2c070e5c545c9858d03ecfb744bf8d717717efc	
4	a	p – 3	"id-GostR3410-2001-CryptoPro-C-ParamSet", "1 2 643 2 2 35 3", "1.2.643.2.2.35.3", "C", "3", "id-GostR3410-2001-CryptoPro-XchB-Param-Set", "1.2.643.2.2.36.1", "1 2 643 2 2 36 1", "XchB".
	b	805a	
	p	9b9f605f5a858107ab1ec85e6b41c8aacf846e86789051d37998f7b9022d759b	
	q	9b9f605f5a858107ab1ec85e6b41c8aa582ca3511eddfb74f02f3a6598980bb9	
	x	0	
	y	41ece55743711a8c3cbf3783cd08c0ee4d4dc440d4641a8f366e550dfdb3bb67	

8. УПРАВЛЕНИЕ КЛЮЧАМИ

Для хранения ключей в криптопровайдере LCJCE используется хранилище в виде файла формата LKS или PKCS12. В JCE хранилище называется *"keystore".*

Приложения имеют доступ к хранилищу ключей через экземпляр класса *KeyStore*. Способы работы с хранилищем даны в описании класса *KeyStore*.

Для работы с хранилищем ключей используется объект, получаемый с помощью метода *getInstance* предоставляемого классом *KeyStore*.

8.1. Класс KeyStore

Класс *KeyStore* определяет интерфейс для доступа к хранилищу ключей и изменения его содержимого. Класс представляет размещенный в памяти набор ключей и сертификатов. В *KeyStore* хранятся объекты двух типов: ключи и доверенные сертификаты. Объекты ключей могут быть сеансовыми симметричными ключами (ГОСТ 28147-89) или закрытыми ключами (ГОСТ Р 34.10-2001) с цепочкой сертификатов открытых ключей, подтверждающих данный закрытый ключ. Объекты ключей хранятся в зашифрованном виде. Доверенные сертификаты, хранимые в хранилище называются так потому, что они являются проверенными, т.е. их подлинность и действительность подтверждена.

Замечание. Хранение доверенных сертификатов в хранилище PKCS12 провайдером LCJCE не поддерживается. Доверенные сертификаты можно хранить в штатном хранилище типа JKS или в хранилище типа LKS, поддерживаемом провайдером LCJCE.

Как и для всех других *engine классов*, объект *KeyStore* получается с помощью одного из методов *getInstance*:

```
static KeyStore getInstance(String algorithm)
static KeyStore getInstance(String algorithm, String provider)
static KeyStore getInstance(String algorithm, Provider provider)
```

Строка `algorithm` должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для класса *KeyStore*.

Прежде, чем использовать объект *KeyStore*, необходимо загрузить данные хранилища. Для загрузки данных хранилища в память используется метод *load*:

```
final void load(InputStream stream, char[] password)
```

Каждому объекту в хранилище присвоен идентификатор. Для получения списка идентификаторов используется метод *aliases*:

```
final Enumeration aliases()
```

Для определения типа объекта используются методы:

```
final boolean isKeyEntry(String alias)
final boolean isCertificateEntry(String alias)
```

Методы добавления получения и удаления объектов:

```
final void setCertificateEntry(String alias, Certificate cert)
final void setKeyEntry(String alias, Key key,  char[] password,
                       Certificate[] chain)
final void setKeyEntry(String alias, byte[] key,
                       Certificate[] chain)
final void deleteEntry(String alias)
final Key getKey(String alias, char[] password)
final Certificate getCertificate(String alias)
final Certificate[] getCertificateChain(String alias)
final String getCertificateAlias(Certificate cert)
```

Сохранение *KeyStore*

```
final void store(OutputStream stream, char[] password)
```

9. КЛАССЫ КРИПТОГРАФИЧЕСКИХ ФУНКЦИЙ

9.1. Класс MessageDigest

Класс *MessageDigest* является *механизмом,* предназначенным для вычисления хэш функции (дайджеста) сообщения. Для вычисления хэш функции по ГОСТ Р 34.11-94, с использованием криптопровайдера LCJCE необходимо:

- Создать объект класса *MessageDigest*.

Для этого вызывается один из методов *getInstance* класса *MessageDigest*:

```
static MessageDigest getInstance(String algorithm)
static MessageDigest getInstance(String algorithm, String pro-
vider)
static MessageDigest getInstance(String algorithm, Provider pro-
vider)
```

Строка *algorithm* должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для алгоритма ГОСТ 34.11-94.

Вызов *getInstance* возвращает инициализированный объект для вычисления хэш функции. Дальнейшая инициализация объекта *MessageDigest* ненужна.

- Вычислить хэш от некоторого объема данных путем вызова одного или нескольких последовательных методов *update*:

```
void update(byte input)
void update(byte[] input)
void update(byte[] input, int offset, int len)
```

- Завершить вычисление с помощью метода *digest*:

```
byte[] digest()
byte[] digest(byte[] input)
int digest(byte[] buf, int offset, int len)
```

Первые два метода возвращают результат вычисления хэша, а последний помещает хэш в буфер *buf*, начиная с позиции *offset*. Последний метод возвращает длину хэша.

Пример:

```
MessageDigest h = MessageDigest.getInstance("HASH");
System.out.println("MessageDigest algorithm name is " +
                   h.getAlgorithm());
h.update(new byte[] { 0, 1, 2, 3, 4, 5, 6, 7 });
byte[] hb = h.digest();


System.out.println (
   "HASH: " +
ru.lc.security.crypto.CryptoUtils.toStringBlock( hb ) );
```

9.2. Класс Signature

Класс *Signature* является *механизмом,* предназначенным для вычисления и проверки ЭЦП. Объекты класса Signature являются модальными. Это значит, что они могут находиться в определенном состоянии. Состояние определяется внутренней константой объекта, которая может иметь следующие значения:

```
UNINITIALIZED
SIGN
VERIFY
```

Для выработки и проверки ЭЦП по ГОСТ Р 34.10-2001, с использованием криптопровайдера LCJCE необходимо:

- Создать объект `Signature` с помощью одного из методов:

```
static Signature getInstance(String algorithm)
static Signature getInstance(String algorithm, String provider)
static Signature getInstance(String algorithm, Provider pro-
vider)
```

Строка `algorithm` должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для алгоритма ЭЦП (ГОСТ Р 34.10-2001).

- Инициализировать объект *Signature* для подписи:

```
final void initSign(PrivateKey privateKey)
```

или для проверки подписи:

```
final void initVerify(PublicKey publicKey)
final void initVerify(Certificate certificate)
```

- Передать подписываемые данные в объект с помощью одного или нескольких вызовов методов:

```
final void update(byte b)
final void update(byte[] data)
final void update(byte[] data, int off, int len)
```

- Если объект был проинициализирован для подписи, то сгенерировать подпись:

```
final byte[] sign()
final int sign(byte[] outbuf, int offset, int len)
```

вызов метода *sign* сбрасывает состояние объекта в то, которое было после вызова метода *initSign*. В этом состоянии возможны вызовы *update*, *initSign* и *initVerify*.

- Если объект был проинициализирован для проверки подписи, то проверить подпись:

```
final boolean verify(byte[] signature)
final boolean verify(byte[] signature, int offset, int length)
```

вызов метода `verify` сбрасывает состояние объекта в то, которое было после вызова метода *initSign*. В этом состоянии возможны вызовы *update*, *initSign* и *initVerify*.

9.3. Класс SecureRandom

Класс *SecureRandom* является *механизмом*, который обеспечивает выработку последовательностей случайных чисел (ПСЧ).

Как и для всех других *механизмов*, объект *SecureRandom* получается с помощью одного из методов *getInstance*.

```
static SecureRandom getInstance(String algorithm)
static final SecureRandom getInstance(String algorithm, String
provider)
static final SecureRandom getInstance(String algorithm, Provider
provider)
```

Строка *algorithm* должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для класса *SecureRandom*.

Для выработки ПСЧ используется алгоритм ГОСТ 28147-89, работающий в режиме гаммирования. ПСЧ представляет собой гамму, выработанную на некотрорм ключе и синхропосылке. Первоначально 32 байта ключа и 8 байт синхропосылки (всего 40 байт) получаются с помощью системного ДСЧ.

Для повышения случайности ПСЧ, можно установить начальное состояния генератора случайных чисел с помощью методов *setSeed*:

```
synchronized public void setSeed(byte[] seed)
public void setSeed(long seed)
```

Рекомендуется использовать первый метод: *setSeed(byte[] seed)*, при этом размер массива *seed* должен быть 40 байт. Полученный массив складывается побайтно с помощью операции XOR со старым и результат используется как ключ и синхропосылка.

После того, как был вызван *setSeed* метод на объекте *SecureRandom* вырабатываемая случайная последовательность будет настолько случайной, насколько были случайны параметры метода *setSeed*.

После выработки 1 килобайта ПСЧ производится переинициализация генератора с использованием дополнительно выработанных 40 байт ПСЧ.

Для получения ПСЧ используется метод *nextBytes*:

```
synchronized public void nextBytes(byte[] bytes)
```

Также ПСЧ можно получать с помощью метода generateSeed:

```
public byte[] generateSeed(int numBytes)
```

9.4. Класс Cipher

Класс *Cipher* обеспечивает функции шифрования. Класс *Cipher* является *engine классом*, и как для всех других *engine классов*, объект *Cipher* получается с помощью одного из методов *getInstance*:

```
public static Cipher getInstance(String transformation);
public static Cipher getInstance(String transformation,
                                 String provider);
```

Значение параметра *transformation* должно принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для класса *Cipher*, а также может включать дополнительные поля, определяющие режим работы объекта *Cipher*.

Значение параметра *transformation* может определяться строкой одного из двух видов:

"algorithm/mode/padding"

"algorithm"

В первом случае дополнительно к имени идентификатору алгоритма из таблицы Таблица. 1 для класса *Cipher*, должны также указываться режим шифрования (mode) и режим дополнения (padding – режим выравнивания данных на границу блока шифрования).

Для реализации шифратора криптопровайдера *LCJCE* допустимыми идентификаторами режимов шифрования являются: *"ECB"* (простая замена), *"CNT" или "GAM"* (гаммирование), *"CFB" или "OFB"* (гаммирование с обратной связью) и *"CBC"* (сцепление блоков).

Допустимыми идентификаторами пэддинга являются: *"NO_PAD"* (нет пэддинга), *"ZERO_PAD"* (дополнение нулями), *"PKCS5_PAD"* (пэддинг в соответствии PKCS#5). Замечание: выравнивание зашифрованных данных на границу блока (пэддинг) необходимо только для шифрования в режимах *"ECB"* и *"CBC"*.

Заметим, что при расшифровке данных, которые были зашифрованы с пэддингом *"ZERO_PAD"*, добавленные нулевые байты в конце расшифрованного текста не удаляются, поскольку считается, что их количество может знать наверняка только приложение более высокого уровня.

Примером допустимого значения параметра transformation метода getInstance для провайдера LCJCE может быть:

"GOST/ECB/PKCS5_PAD"

или

"GOST/GAM/NO_PAD"

Объект *Cipher* после вызова метода *getInstance* необходимо инициализировать. В процессе инициализации устанавливается режим (шифрование/расшифровка), ключ, параметры и источник случайных чисел объекта

Cipher. Инициализация устанавливает объект *Cipher* в один из четырех режимов, которые определены константами класса *Cipher*:

ENCRYPT_MODE – режим шифрования;

DECRYPT_MODE – режим расшифровки;

WRAP_MODE – режим шифрования ключа;

UNWRAP_MODE – режим расшифровки ключа.

Провайдер *LirJCE* не поддерживает два последних режима.

Для инициализации объекта *Cipher* используется один из методов *init*:

```
    public void init(int opmode, Key key);
    public void init(int opmode, Certificate certificate)
    public void init(int opmode, Key key, SecureRandom random);
    public void init(int opmode, Certificate certificate, Se-
cureRandomrandom)
    public void init(int opmode, Key key, AlgorithmParameterSpec
params);
    public void init(int opmode, Key key, AlgorithmParameterSpec
params,
                     SecureRandom random);
    public void init(int opmode, Key key, AlgorithmParameters
params)
    public void init(int opmode, Key key, AlgorithmParameters
params,
                     SecureRandom random)
```

Параметр *key* должен являться объектом *Gost_28147Key*, полученным, например, с помощью объекта *KeyGenerator* криптопровайдера LirJCE (см. ниже).

Шифрование и расшифровка данных с помощью проинициализированного объекта *Cipher* может осуществляться за одну или несколько операций. Несколько операций используются в том случае, если сразу неизвестно сколько данных необходимо обработать.

Для шифрования или расшифровки данных за один шаг необходимо использовать один из методов *doFinal*:

```
    public byte[] doFinal(byte[] input);
    public byte[] doFinal(byte[] input, int inputOffset, int in-
putLen);
    public int doFinal(byte[] input, int inputOffset,
                       int inputLen, byte[] output);
```

```
    public int doFinal(byte[] input, int inputOffset,
                       int inputLen, byte[] output, int out-
putOffset)
```

Для шифрования или расшифровки данных за несколько шагов необходимо использовать один из методов *update*:

```
    public byte[] update(byte[] input);
    public byte[] update(byte[] input, int inputOffset, int in-
putLen);
    public int update(byte[] input, int inputOffset, int input-
Len,
                      byte[] output);
    public int update(byte[] input, int inputOffset, int input-
Len,
                      byte[] output, int outputOffset)
```

Несколько операций шифрования или расшифровки должны завершаться одним из методов *doFinal*, которые были представлены выше.

Выполнение метода *doFinal* сбрасывает объект *Cipher* в то состояние, которое у него было после вызова метода *init*.

9.4.1. Управление параметрами

Для алгоритма ГОСТ 28147-89, реализованного в криптопровайдере LCJCE, криптопараметрами являются узлы замен (подстановок), представляющие собой массив из 64 байт, вектор инициализации (синхропосылка 8 байт), а также флаг установки режима модификации ключа через каждый килобайт данных. В общем случае параметры передаются в метод *init* посредством класса *Gost_28147ParameterSpec*, являющегося реализацией интерфейса *AlgorithmParametersSpec* (см. Класс Gost_28147ParameterSpec). Если для инициализации используется метод *init* без параметра *params*, то используется узел замен *"gost28147-89-UZ-CryptoPro-A"*, *OID 1.2.643.2.2.31.1* (см. Таблица. 5) и нулевой вектор инициализации ({0,0,0,0,0,0,0,0}).

Используемые параметры могут быть получены из объекта *Cipher* с помощью метода *getParameters*. Параметры возвращаются в виде объекта *java.security.AlgorithmParameters*.

Особенностью объекта *Cipher* является то, что вектор инициализации (*IV*) может быть передан в объект только в качестве параметра *params* метода *init*. Для обеспечения возможности установки узла замен, *IV* и режима модификации ключа в качестве параметра *params* метода *init* необходимо использовать

объект класса *Gost_28147ParameterSpec*. Кроме того, для установки только вектора инициализации можно использовать в качестве параметра *params* метода *init* объект класса *IvParameterSpec*. Надо заметить, что первоначально установленный вектор *IV* можно получить из объекта *Cipher* с помощью метода *getIV*.

Пример:

```
import javax.crypto.*;
import java.security.AlgorithmParameters;
import ru.lc.security.crypto.*;

// вырабатываем ключ
KeyGenerator keygen = KeyGenerator.getInstance("GOST");
SecretKey myKey = keygen.generateKey();

// получаем объект Cipher в режиме гаммирования
javax.crypto.Cipher c = Cipher.getInstance("GOST/GAM/NO_PAD");

// делаем криптопараметр - таблицу замен
AlgorithmParameterSpec parSpec = new Gost_28147ParameterSpec(
new byte[] {
   (byte)0x43,(byte)0x62,(byte)0x23,(byte)0x11,
   (byte)0x6A,(byte)0x7A,(byte)0x5E,(byte)0x85,
   (byte)0x9F,(byte)0xB7,(byte)0x9C,(byte)0x0F,
   (byte)0xFE,(byte)0x08,(byte)0x8F,(byte)0xB2,
   (byte)0xEC,(byte)0x9C,(byte)0xD8,(byte)0x79,
   (byte)0xC6,(byte)0xCB,(byte)0x49,(byte)0xC6,
   (byte)0x0D,(byte)0x3E,(byte)0x10,(byte)0x28,
   (byte)0xBB,(byte)0x2F,(byte)0xF7,(byte)0x44,
   (byte)0x21,(byte)0xD6,(byte)0xC1,(byte)0x93,
   (byte)0x38,(byte)0x4D,(byte)0x6A,(byte)0xDA,
   (byte)0x19,(byte)0xA9,(byte)0xB2,(byte)0xEB,
   (byte)0x85,(byte)0x80,(byte)0x06,(byte)0x6C,
   (byte)0xD4,(byte)0xE3,(byte)0xAD,(byte)0x57,
   (byte)0x50,(byte)0x11,(byte)0x74,(byte)0xAD,
   (byte)0x77,(byte)0x55,(byte)0xEB,(byte)0x3E,
   (byte)0xA2,(byte)0xF4,(byte)0x35,(byte)0xF0
```

```
  });


  // делаем криптопараметр - синхропосылку
  AlgorithmParameterSpec parameterSpec =
       new IvParameterSpec(new byte[] { 0, 0, 0, 0, 0, 0, 0,
0 });


  // инициализируем объект Cipher для шифрования с установкой
  // таблицы замен
  c.init(javax.crypto.Cipher.ENCRYPT_MODE, myKey, parSpec);


  // инициализируем объект Cipher для шифрования с установкой
  // синхропосылки
  c.init(javax.crypto.Cipher.ENCRYPT_MODE, myKey,
parameterSpec);


  // печатаем исходный текст
  byte[] cleartext = "This is just an example".getBytes();
  System.out.println("cleartext = " +
       CryptoUtils.toStringBlock(cleartext));


  // шифруем строку исходного текста
  byte[] cipherText = c.doFinal("This is just an example".get-
Bytes());


  // Инициализируем объект Cipher для расшифровки
  // Инициализировать таблицу замен не нужно, так как данный
объект
  // Cipher уже имеет нужную таблицу замен
  c.init(javax.crypto.Cipher.DECRYPT_MODE, myKey,
parameterSpec);


  // дешифруем текст
  byte[] cleartext1 = gostCipher.doFinal(ciphertext);
  System.out.println ("cleartext1 = " +
       CryptoUtils.toStringBlock(cleartext1));
```

В пакете *ru.lc.security.crypto* содержится класс *CryptoProParam*, который позволяет получить объект *Gost_28147ParameterSpec* для инициализа-

ции узла замен объекта *Cipher* криптопровайдера *LCJCE*, который позволяет использовать символьные имена и идентификаторы объектов (OID), определенные фирмой Крипто Про следующим образом :

```
 Gost_28147ParameterSpec parSpec =
CryptoProParam.get28147Par("A");
```

В таблице Таблица. 5 приведены узлы замен и соответствующие им символьные идентификаторы (*id*) для метода *CryptoProParam.get28147Par(String id)*:

Таблица. 5

№	Узел замен	Символьные идентификаторы
1	4E5764D1AB8DCBBF941A7A4D2CD11010 D6A057358D38F2F70F49D15AEA2F8D94 62EE4309B3F4A6A218C698E3C17CE57E 706B0966F7023C8B5595BF2839B32ECC	"id-GostR3411-94-TestParamSet", "1 2 643 2 2 30 0", "1.2.643.2.2.30.0", "0", "test"
2	A57477D14FFA66E354C7424A60ECB419 82909D751D4FC90B3B122F547908A0AF D13E1A38C7B181C6E65605870325EBFE 9C6DF86D2EABDE20BA893C92F8D353BC	"id-GostR3411-94-CryptoProParamSet", "1 2 643 2 2 30 1", "1.2.643.2.2.30.1", "1.2.643.2.2.9", "1"
3	93EEB31B67475ADA3E6A1D2F292C9C95 88BD8170BA31D2AC1FD3F06E70890B08 A5C0E78642F245C2E65B2943FCA43459 CB0FC8F104787F37DD15AEBD519666E4	"gost28147-89-UZ-CryptoPro-A", "UZ_A", "28147_1", "A"
4	80E7285041C57324B200C2AB1AADF6BE 349B94985D265D1305D1AEC79CB2BB31	"gost28147-89-UZ-CryptoPro-B", "UZ_B", "28147_2",

	29731C7AE75A4142A38C07D9CFFFDF06 DB346A6F686E80FD7619E985FE4835EC	"B"
5	10838CA7B126D994C750BB602D010185 9B4548DAD49D5EE205FA122FF2A8240E 483B97FC5E7233368FC9C651ECD7E5BB A96E6A4D7AEFF019661CAFC333B47D78	"gost28147-89-UZ-CryptoPro-C", "UZ_C", "28147_3", "C"
6	FB110831C6C5C00A23BE8F66A40C93F8 6CFAD21F4FE725EB5E60AE90025DBB24 77A671DC9DD23A83E84B64C5D0845749 15994CB7BA33E9AD897FFD523128167E	"gost28147-89-UZ-CryptoPro-D", "UZ_D", "28147_4", "D"

Некоторые методы *update* и *doFinal* объекта *Cipher* позволяют определить буфер, в который должны помещаться зашифрованные или расшифрованные данные. В таких случаях важно передать буфер достаточный для размещения результата. Следующий метод объекта *Cipher* позволяет определить требуемый размер буфера:

```
public int getOutputSize(int inputLen);
```

9.5. Класс KeyGenerator

Класс *KeyGenerator* является *механизмом*, который обеспечивает выработку ключей шифрования.

Как и для всех других *механизмов*, объект *KeyGenerator* получается с помощью одного из методов *getInstance*.

```
public static KeyGenerator getInstance(String algorithm);
public static KeyGenerator getInstance(String algorithm,
                                       String provider);
```

Строка *algorithm* должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы …» таблицы Таблица. 1 для класса *KeyGenerator*.

Существуют три метода инициализации объекта *KeyGenerator*:

```
  public void init(SecureRandom random);
  public void init(int keysize);
  public void init(int keysize, SecureRandom random);
```

Криптопровайдер LCJCE не учитывает значение параметра *keysize*, так как размер ключа для ГОСТ 28147-89 всегда равен 256 бит.

Рекомендуется использовать для инициализации объекта *KeyGenerator* метод *init* с параметром *random*, который является объектом *SecureRandom* криптопровайдера LCJCE.

Для выработки ключа шифрования используется метод *generateKey*:

```
  public SecretKey generateKey();
```

9.6. Класс Mac

Класс *Mac* обеспечивает функции выработки кода аутентификации сообщения (Message Authentication Code).

Класс *Mac* является *механизмом*. Как и для всех других *механизмов*, объект *Mac* получается с помощью одного из методов *getInstance*.

```
  public static Mac getInstance(String algorithm);
  public static Mac getInstance(String algorithm, String pro-
vider);
```

Строка *algorithm* должна принимать одно из значений из определенных в колонке «Допустимые идентификаторы» таблицы Таблица. 1 для класса *Mac*.

Объект должен быть инициализирован с помощью метода *init*:

```
  public void init(Key key, AlgorithmParameterSpec params);
```

Для инициализации объекта *Mac,* параметр *key* должен являться объектом *Gost_28147Key*, полученным, например, с помощью объекта *KeyGenerator* криптопровайдера LCJCE (см. выше). Параметр *params* должен соответствовать требованиям, определенным в разделе «Класс *Cipher*».

Код аутентификации сообщения (MAC) может быть вычислен за один или несколько шагов. Несколько шагов используются в том случае, если сразу неизвестно сколько данных необходимо обработать.

Для вычисления MAC за один шаг используется функция *doFinal*:

```
  public byte[] doFinal(byte[] input);
```

Для вычисления MAC за несколько шагов вызываются методы *update*:

```
  public void update(byte input);
  public void update(byte[] input);
  public void update(byte[] input, int inputOffset, int input-
Len);
```

Многошаговая операция вычисления MAC должна заканчиваться вызовом метода *doFinal*, который был описан выше, если еще остались входные данные или следующими методами *doFinal*, если данных для вычисления MAC больше нет:

```
  public byte[] doFinal();
  public void doFinal(byte[] output, int outOffset);
```

Класс *Mac* криптопровайдера LCJCE является клонируемым (cloneable), т.е. до вызова метода *doFinal* можно создать дубликат объекта MAC. Один из полученных идентичных MAC объектов можно использовать для получения промежуточного значения MAC, а другой для продолжения вычисления MAC для следующих порций входных данных.

10. ПРИЛОЖЕНИЕ А: СПЕЦИФИКАЦИЯ РЕАЛИЗОВАННЫХ МЕХАНИЗМОВ

10.1. MessageDigest

Реализовано вычисление хэш в соответствии с ГОСТ Р 34.11-94. Реализация является клонируемой, т.е. может быть создана точная копия объекта. Имя алгоритма: GOST3411. Допустимыми именами являются также: HASH, GOST, GOST_HASH, HASH_34_11_94. OID алгоритма – 1.2.643.2.2.9. Функция вычисления хэша, поддерживаемая провайдером LCJCE, вырабатывает значение хэша как массив из 32 байт.

10.2. Signature

Реализована выработка и проверка ЭЦП в соответствии с ГОСТ Р 34.10-2001. Имя алгоритма GOST3411withGOST34102001. Допустимые имена: GOST, GOST34102001, GOST_2001, GOST_34_10_01. OID алгоритма – 1.2.643.2.2.3. Функция вычисления ЭЦП, поддерживаемая провайдером LCJCE, вырабатывает значение ЭЦП как массив из 64 байт.

10.3. SecureRandom

Реализован высококачественный алгоритм выработки случайного числа на основе ГОСТ 28147-89 и случайного начального заполнения с периодической перезагрузкой. Имя алгоритма GOST. Допустимые имена: GOST, GOST_RNG. OID алгоритма – 1.2.643.2.18.2.1.

10.4. KeyPairGenerator

Выработка пар ключей ЭЦП в соответствии с ГОСТ Р 34.10-2001. Имя алгоритма GOST34102001. Допустимые имена: GOST, GDS_2001. OID алгоритма – 1.2.643.2.2.19. Необходима предварительная инициализация параметров ЭЦП (см. Класс KeyPairGenerator).

10.5. AlgorithmParameters

Реализована работа с параметрами алгоритма ЭЦП в соответствии с ГОСТ Р 34.10-2001. Имя алгоритма GOST34102001. Допустимое имя: GOST_DS. OID алгоритма – 1.2.643.2.2.19. Спецификация параметров ЭЦП, поддерживаемая провайдером LCJCE, определена в интерфейсе Gost_DSParams:

```
/*
 * Gost_DSParams.java
 *
 */

package ru.lc.security.crypto.interfaces;
import java.math.*;

public interface Gost_DSParams {

    public static String ALGNAME = "GOST34102001";

    public BigInteger getA();
    public BigInteger getB();
    public BigInteger getP();
    public BigInteger getQ();
    public BigInteger getX();
    public BigInteger getY();
    public String getOID();
    public void setOID(String dsOID);

    public Gost_28147Params getCipherPar();

}
```

Реализована работа с параметрами алгоритма шифрования в соответствии с ГОСТ 28147-89. Имя алгоритма GOST28147. Допустимые имена GOST28147, CIPHER. OID алгоритма – 1.2.643.2.2.21. Спецификация параметров шифрования, поддерживаемая провайдером LCJCE, определена в интерфейсе Gost_28147Params:

```
/*
 * Gost_28147Params.java
 *
 */

package ru.lc.security.crypto.interfaces;

public interface Gost_28147Params {
    public static String ALGNAME = "GOST28147";
    /**
     * Получение OIDa параметра
     */
    public String getOID();
    /**
     * Установка OIDa параметра
     */
    public void setOID(String hashOID);
    public byte[] getTZ();
}
```

10.6. KeyFactory

Имя алгоритма GOST34102001. Допустимые имена: GOST, GOST_DS. OID алгоритма – 1.2.643.2.2.19. Key Factories, поддерживаемая провайдером LCJCE, обеспечивает работу с ключам ЭЦП и параметрами по ГОСТ Р 34.10-2001.

10.7. KeyStore

Реализованы защищенные хранилища ключей и сертификатов в форматах LKS и PKCS12. Имена алгоритмов: LKS, PKCS12. Защищенное хранилище, поддерживаемое провайдером LCJCE, обеспечивает защищенное хранение ключей ЭЦП ГОСТ 34.10-2001, сертификатов X509 открытых ключей, сеансовых ключей. Хранение доверенных сертификатов в хранилище PKCS12 не поддерживается провайдером LCJCE.

10.8. Cipher

Реализован алгоритм шифрования в соответствии с ГОСТ 28147-89. Имя алгоритма GOST28147. Допустимые имена: GOST, GOST_28147, CIPHER. OID алгоритма – 1.2.643.2.2.21. Функция шифрования, поддерживаемая провайдером LCJCE, обеспечивает шифрование в соответствии с ГОСТ 28147-89 во всех определенных режимах.

10.9. MAC - Имитовставка

Реализовано вычисление имитовставки в соответствии с ГОСТ 28147-89. Имя алгоритма: GOST28147. Допустимыми именами являются также: GOST, IMIT, Gost28147-MAC. OID алгоритма – не определен. Функция вычисления имитовставки, поддерживаемая провайдером LCJCE, вырабатывает значение MAC как массив из 4 байтов.

10.10. KeyGenerator

Выработка ключей шифрования в соответствии с ГОСТ 28147. Имя алгоритма GOST28147. Допустимые имена: GOST, GOST_28147, Generic. OID алгоритма – не определен. Необходима предварительная инициализация параметров (см. раздел Параметры). Функция выработки ключей шифрования, поддерживаемая провайдером LCJCE, вырабатывает ключ как массив байт.

10.11. KeyAgreement

Выработка ключей согласования в соответствии с RFC 3257. Имя алгоритма: GOST34102001. Допустимые имена: GOST, GOST_DH. Необходима предварительная инициализация параметров (см. раздел Параметры). Функция выработки ключей согласования, поддерживаемая провайдером LCJCE, вырабатывает ключ как массив 32 байт.

10.12. HMAC

Реализовано вычисление HMAC в соответствии с RFC 3257. Имя алгоритма: GOST3411-HMAC. Допустимыми именами являются также: HMAC, HMAC_GOST, HMAC_GOST3411. OID алгоритма – 1.2.643.2.2.10. Функция вычисления HMAC, поддерживаемая провайдером LCJCE, вырабатывает значение HMAC как массив из 32 байт. Алгоритм инициализируется ключом типа Generic, размер которого может отличаться от 32-х байтов.