127 018, Москва, Сущевский Вал, 18
Телефон: (495) 995 4820
Факс: (495) 995 4820
http://www.CryptoPro.ru
E-mail: info@CryptoPro.ru

Средство Криптографической Защиты Информации

КриптоПро JCP

Версия 2.0

Руководство программиста. Модули шифрования

ЖТЯИ.00091-01 33 02
Листов 21

2016 г.

ЖТЯИ.00091-01 33 02. КриптоПро JCP. Руководство программиста. Модули шифрования.

Оглавление

1. Введение

Настоящее руководство содержит описание основной функциональности криптопровайдера «КриптоПро JCP» версия 2.0 (модули шифрования) и примеры их использования.

Криптопровайдер «КриптоПро JCP» версия 2.0 является средством криптографической защиты информации (СКЗИ «КриптоПро JCP» версия 2.0), реализующим российские криптографические алгоритмы и функционирующим под управлением виртуальной машины Java 2 Runtime Environment версии 1.6 и выше, соответствующей спецификации Sun Java 2 ™ Virtual Machine.

Криптопровайдер «КриптоПро JCP» версия 2.0 (модули шифрования) реализует стандартный интерфейс Java Cryptography Extension (JCE) в соответствии с российскими криптографическими алгоритмами и обеспечивает выполнение следующих операций через стандартный интерфейс JCE:

- генерация закрытых (256 бит) и открытых (512 бит) ключей обмена в соответствии с алгоритмами подписи ГОСТ Р 34.10-2001 и обмена Диффи-Хелмана с различными параметрами;
- генерация закрытых и открытых ключей обмена в соответствии с алгоритмами подписи ГОСТ Р 34.10-2012 и обмена Диффи-Хелмана с различными параметрами;
- выработка ключей согласования в соответствии с алгоритмом обмена Диффи-Хелмана;
- генерация симметричных ключей шифрования, соответствующих алгоритму ГОСТ 28147-89;
- шифрование данных во всех режимах, определенных ГОСТ 28147-89;
- шифрование ключей в соответствии с алгоритмом ГОСТ 28147-89;
- имитозащита данных в соответствии с алгоритмом ГОСТ 28147-89;

Основные технические данные и характеристики СКЗИ, а также информацию о совместимости с другими продуктами КриптоПро см. в «Руководстве администратора безопасности».

2. Работа с ключами в криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования)

Основная работа с ключами в криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования) осуществляется в соответствии с интерфейсами JCE. К ним относятся выработка ключей согласования по алгоритму Диффи-Хелмана, генерация симметричных ключей шифрования. Но существуют также методы, работающие с ключами через интерфейс JCA. К ним относятся генерация ключевых пар обмена, соответствующих алгоритмам обмена Диффи-Хелмана и подписи ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012 (256).

2.1. Работа с ключевыми парами (закрытый и открытый ключи) обмена, соответствующими алгоритмам обмена Диффи-Хелмана и подписи ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012

Работа с ключевыми парами обмена в криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования) по большей своей части осуществляется через стандартный интерфейс JCA при помощи криптопровайдера «КриптоПро JCP» версия 2.0, описание которого приводится в «Руководстве программиста КриптоПро JCP». Основной класс провайдера «КриптоПро JCP» версия 2.0 (модули шифрования) - ru.CryptoPro.Crypto.CryptoProvider.

Небольшие отличия в использовании интерфейса JCA в криптопровайдерах «КриптоПро JCP» версия 2.0 и «КриптоПро JCP» версия 2.0 (модули шифрования) возникают в процессе генерации ключевой пары:

1. Первое отличие возникает в процессе создания генератора ключевой пары. Для генерации ключевой пары обмена, соответствующей алгоритмам обмена Диффи-Хелмана и подписи ГОСТ Р 34.10-2001, необходимо при создании генератора в качестве имени, идентифицирующего требуемые алгоритмы, указывать имя "GOST3410DHEL" или JCP.GOST_EL_DH_NAME. А в случае генерации ключевой пары обмена, соответствующей алгоритмам обмена Диффи-Хелмана и подписи ГОСТ Р 34.10-2012 (256) или ГОСТ Р 34.10-2012 (512), указывать имя "GOST3410DH_2012_256" или JCP.GOST_DH_2012_256_NAME, или "GOST3410DH_2012_512" или JCP.GOST_DH_2012_512_NAME. Таким образом, в криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования) генератор ключевой пары обмена, соответствующей алгоритмам обмена Диффи-Хелмана и подписи ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012 производится одним из следующих способов:

```
KeyPairGenerator kg = KeyPairGenerator.getInstance("GOST3410DHEL");

KeyPairGenerator kg = KeyPairGenerator.getInstance("GOST3410DHEL", "Crypto");

KeyPairGenerator kg = KeyPairGenerator.getInstance(JCP.GOST_EL_DH_NAME,
CryptoProvider.PROVIDER_NAME);
```

```
KeyPairGenerator kg = KeyPairGenerator.getInstance("GOST3410DH_2012_256");

KeyPairGenerator kg = KeyPairGenerator.getInstance("GOST3410DH_2012_256",
"Crypto");

KeyPairGenerator kg = KeyPairGenerator.getInstance(JCP.GOST_DH_2012_256_NAME,
CryptoProvider.PROVIDER_NAME);
```

```
KeyPairGenerator kg = KeyPairGenerator.getInstance("GOST3410DH_2012_512");

KeyPairGenerator kg = KeyPairGenerator.getInstance("GOST3410DH_2012_512",
"Crypto");

KeyPairGenerator kg = KeyPairGenerator.getInstance(JCP.GOST_DH_2012_512_NAME,
CryptoProvider.PROVIDER_NAME);
```

2. Второе отличие возникает в процессе изменения параметров генератора ключевой пары. Также, как и в криптопровайдере «КриптоПро JCP» версия 2.0, генератор осуществляет создание ключевой пары обмена с параметрами, установленными в контрольной панели (с параметрами по умолчанию). Для того, чтобы изменить набор параметров, в соответствии с которым будет осуществляться создание ключевой пары обмена, следует воспользоваться описанным в документации интерфейсом набора параметров для генерации ключевой пары AlgIdInterface. В криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования) в этот набор параметров входят:

- идентификатор набора параметров для генерации ключевой пары;
- параметры алгоритма обмена Диффи-Хелмана;
- параметры алгоритма хэширования ГОСТ Р 34.11-94 или ГОСТ Р 34.11-2012;
- параметры алгоритма шифрования ГОСТ 28147-89.

Наборы параметров для криптопровайдеров «КриптоПро JCP» версия 2.0 (генерация ключевой пары подписи) и «КриптоПро JCP» версия 2.0 (модули шифрования) (генерация ключевой пары обмена) различаются двумя элементами:

- для криптопровайдера «КриптоПро JCP» версия 2.0 идентификатор набора параметров для генерации ключевой пары определяет создаваемую ключевую пару, как пару подписи. Для криптопровайдера «КриптоПро JCP» версия 2.0 (модули шифрования) этот идентификатор определяет пару обмена. В обоих случаях допустимо создание пары лишь в соответствии с алгоритмами ЭЦП ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012 (256). На данный момент в каждом криптопровайдере существуют единственные идентификаторы, удовлетворяющие этим требованиям: "1.2.643.2.2.19" (JCP.GOST_EL_KEY_OID), "1.2.643.7.1.1.1.1" (JCP.GOST_PARAMS_SIG_2012_256_KEY_OID) и "1.2.643.7.1.1.1.2" (JCP.GOST_PARAMS_SIG_2012_512_KEY_OID) - для «КриптоПро JCP» версия 2.0, "1.2.643.2.2.98" (CryptoProvider.GOST_EL_DH_OID), "1.2.643.7.1.1.6.1" (JCP.GOST_PARAMS_EXC_2012_256_KEY_OID) и "1.2.643.7.1.1.6.2" (JCP.

GOST_PARAMS_EXC_2012_512_KEY_OID) - для «КриптоПро JCP» версия 2.0 (модули шифрования);

- для криптопровайдера «КриптоПро JCP» версия 2.0 вторым элементом набора параметров являются параметры подписи ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012. Для криптопровайдера «КриптоПро JCP» версия 2.0 (модули шифрования) таким элементом являются параметры обмена Диффи-Хелмана, соответствующие алгоритму ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012. Описание изменения таких параметров полностью совпадает с описанием изменения параметров подписи. Единственным отличием является то, что вместо статического метода *getDefaultSignParams()* класса AlgIdSpec, получающего ссылку на интерфейс параметров подписи по умолчанию, следует пользоваться методом *getDefaultExchangeParams()* того же класса. Этот метод возвращает ссылку на интерфейс ParamsInterface параметров обмена по умолчанию (установленных в контрольной панели):

    - ParamsInterface exchParams = AlgIdSpec.getDefaultExchangeParams();

Остальные действия могут быть проведены аналогичным с параметрами подписи образом.

К созданным таким способам ключевым парам обмена могут быть применены все операции, реализуемые криптопровайдером «КриптоПро JCP» версия 2.0 (таких как хранение на носителях закрытых ключей и соответствующих им сертификатов, создание ЭП по алгоритму ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012, генерация сертификатов и так далее). Подробное описание реализаций всех операций приводится в «Руководстве программиста» «КриптоПро JCP» версия 2.0.

Но помимо этих операций, ключевые пары обмена могут также участвовать в формировании ключей согласования, что неверно для ключей ЭП, созданных при помощи криптопровайдера «КриптоПро JCP» версия 2.0. Таким образом, в рамках криптопровайдера «КриптоПро JCP» версия 2.0 функциональные возможности ключевых пар подписи и обмена одинаковы. В рамках же криптопровайдера «КриптоПро JCP» версия 2.0 (модули шифрования) имеет смысл использование только ключевых пар обмена.

2.2. Работа с ключами согласования в соответствии с алгоритмом обмена Диффи-Хелмана

Криптопровайдер «КриптоПро JCP» версия 2.0 (модули шифрования) осуществляет работу с ключами согласования в соответствии с алгоритмом обмена Диффи-Хелмана, через стандартный интерфейс JCE при помощи класса KeyAgreement.

2.2.1. Создание объекта генерации (генератора) ключей согласования

Объект генерации ключей согласования (далее *генератор*) создается посредством вызова метода *getInstance()* класса KeyAgreement. Этот метод является статическим и возвращает ссылку на класс KeyAgreement, который обеспечивает выполнение требуемой операции.

Для создания генератора ключей согласования в соответствии с алгоритмом обмена Диффи-Хелмана методу *getInstance()* необходимо в качестве параметра передать имя, идентифицирующее данный алгоритм ("GOST3410DHEL" или JCP.GOST_EL_DH_NAME и "GOST3410DH_2012_256" или JCP.GOST_DH_2012_256_NAME или "GOST3410DH_2012_512" или JCP.GOST_DH_2012_512_NAME). При таком вызове метода *getInstance()* совместно с определением требуемого алгоритма генерации ключей согласования осуществляется также определение требуемого типа криптопровайдера («КриптоПро JCP» версия 2.0 (модули шифрования)). Также стандартный интерфейс JCE позволяет в качестве параметра функции *getInstance()* класса KeyAgreement вместе с именем алгоритма передавать имя криптопровайдера, используемого для выполнения требуемой операции.

Таким образом, создание генератора ключей обмена в соответствии с алгоритмом Диффи-Хелмана осуществляется одним из следующих способов:

```
KeyAgreement ka = KeyAgreement.getInstance("GOST3410DHEL");
```

```
KeyAgreement ka = KeyAgreement.getInstance("GOST3410DHEL", "Crypto");
```

```
KeyAgreement ka = KeyAgreement.getInstance(JCP.GOST_EL_DH_NAME,
CryptoProvider.PROVIDER_NAME);
```

Для ключей на алгоритме ГОСТ Р 34.10-2012 (256):

```
KeyAgreement ka = KeyAgreement.getInstance("GOST3410DH_2012_256");
```

```
KeyAgreement ka = KeyAgreement.getInstance("GOST3410DH_2012_256", "Crypto");
```

```
KeyAgreement ka = KeyAgreement.getInstance(JCP.GOST_DH_2012_256_NAME,
CryptoProvider.PROVIDER_NAME);
```

Для ключей на алгоритме ГОСТ Р 34.10-2012 (512):

```
KeyAgreement ka = KeyAgreement.getInstance("GOST3410DH_2012_512");
```

```
KeyAgreement ka = KeyAgreement.getInstance("GOST3410DH_2012_512", "Crypto");
```

```
KeyAgreement ka = KeyAgreement.getInstance(JCP.GOST_DH_2012_512_NAME,
CryptoProvider.PROVIDER_NAME);
```

Созданный таким образом генератор осуществляет выработку ключей согласования из ключей обмена, соответствующих алгоритму ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012 по алгоритму Диффи-Хелмана. Ключи обмена, соответствующие алгоритму ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012, могут быть как прочитаны из контейнера, так и получены при помощи генератора ключевой пары обмена (см. выше).

2.2.2.Инициализация генератора ключей согласования

После того, как генератор ключей согласования был создан, необходимо проинициализировать его закрытым ключом обмена, в соответствии с которым будет осуществляться выработка ключей согласования, а также параметрами, на основе которых эта выработка будет производиться. Такая инициализация осуществляется при помощи метода *init(Key key, AlgorithmParameterSpec params)* класса KeyAgreement. Этому методу в качестве параметра `key` передается закрытый ключ обмена, соответствующий алгоритму ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012. В качестве параметров `params`, участвующих в выработке ключей согласования, передается объект класса IvParameterSpec, представляющий собой стартовый вектор для выработки ключей согласования.

Таким образом, инициализация генератора ключей согласования осуществляется:

```
PrivateKey privateKey;  // обязательно закрытый ключ обмена, соответствующий
                        // алгоритму ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012

IvParameterSpec spec;   // стартовый вектор

ka.init(key, spec);
```

2.2.3.Выполнение фазы согласования ключей

После того, как генератор ключей согласования был создан и проинициализирован закрытым ключом и стартовым вектором, требуется проинициализировать его открытым ключом, в соответствии с которым будет сформирован ключ согласования (фаза согласования ключей). Выполнение фазы осуществляется при помощи метода *doPhase(Key key, boolean lastPhase)* класса KeyAgreement. Этому методу в качестве параметра передается открытый ключ, соответствующий алгоритму ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012. Следует помнить, что параметры хэширования и ЭП этого ключа должны совпадать с соответствующими параметрами поданного на инициализацию закрытого ключа. Второй параметр метода *doPhase()* игнорируется. Считается, что эта величина всегда равна `true`.

```
PublicKey publicKey;     // обязательно открытый ключ обмена, соответствующий
                         // закрытому ключу privateKey


ka.doPhase(publicKey, true);
```

Стандартный интерфейс JCE допускает возвращение данным методом объекта класса Key. В криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования) в качестве такого объекта возвращается переданный методу *doPhase()* открытый ключ. Для получения самого ключа согласования необходимо сделать еще один шаг.

2.2.4.Генерация ключа согласования

После того, как определены закрытый и открытый ключи, на основе которых осуществляется выработка ключа согласования, а также параметры (стартовый вектор), участвующие в этой выработке, необходимо сгенерировать собственно ключ согласования. Генерация такого ключа осуществляется при помощи метода `generateSecret(String algorithm)` класса KeyAgreement. В качестве параметра этот метода получает строковое представление алгоритма ключа согласования ("GOST28147" или CryptoProvider.GOST_CIPHER_NAME). Возвращает этот метод объект класса SecretKey, представляющий собой ключ согласования закрытого и открытого ключей обмена, который создан при помощи стартового вектора. Этот ключ является ключом шифрования алгоритма ГОСТ 28147-89 с параметрами шифрования, соответствующими параметрам открытого ключа.

Таким образом, генерация ключа согласования осуществляется:

```
SecretKey agreeKey = ka.generateSecret("GOST28147");


SecretKey agreeKey = ka.generateSecret(CryptoProvider.GOST_CIPHER_NAME);
```

Для выработки 512-битного секретного ключа для использования в функциях HMAC, основанных на использовании хэш-функций ГОСТ Р 34.11-2012 (256 бит) и ГОСТ Р 34.11-2012 (512 бит) следует передать методу строку "SYMMETRIC512" (CryptoProvider .SYMMETRIC_512).

```
SecretKey agreeKey = ka.generateSecret("SYMMETRIC512");
SecretKey agreeKey = ka.generateSecret(CryptoProvider.SYMMETRIC_512);
```

Пример выработки ключей согласования сторонами см. `samples/samples_src.jar/userSamples/CheckImita.java` (входит в комплект поставки программного обеспечения «КриптоПро JCP» версия 2.0).

2.3. Работа с симметричными ключами шифрования, соответствующими алгоритму ГОСТ 28147-89

Криптопровайдер «КриптоПро JCP» версия 2.0 (модули шифрования) осуществляет генерацию симметричных ключей шифрования, через стандартный интерфейс JCE при помощи класса KeyGenerator.

2.3.1. Создание объекта генерации (генератора) симметричных ключей шифрования

Объект генерации симметричных ключей шифрования (далее *генератор*) создается посредством вызова метода *getInstance()* класса KeyGenerator. Этот метод является статическим и возвращает ссылку на класс KeyGenerator, который обеспечивает выполнение требуемой операции.

Для создания генератора симметричных ключей шифрования, соответствующих алгоритму ГОСТ 28147-89 методу *getInstance()* необходимо в качестве параметра передать имя, идентифицирующее данный алгоритм ("GOST28147" или CryptoProvider.GOST_CIPHER_NAME). При таком вызове метода *getInstance()* совместно с определением требуемого алгоритма генерации ключей согласования осуществляется также определение требуемого типа криптопровайдера («КриптоПро JCP» версия 2.0 (модули шифрования)). Также стандартный интерфейс JCE позволяет в качестве параметра функции *getInstance()* класса KeyGenerator вместе с именем алгоритма передавать имя криптопровайдера, используемого для выполнения требуемой операции.

Таким образом, создание генератора симметричных ключей шифрования, соответствующих алгоритму ГОСТ 28147-89 осуществляется одним из следующих способов:

```
KeyGenerator kg = KeyGenerator.getInstance("GOST28147");
```

```
KeyGenerator kg = KeyGenerator.getInstance("GOST28147", "Crypto");
```

```
KeyGenerator kg = KeyGenerator.getInstance(CryptoProvider.GOST_CIPHER_NAME,
CryptoProvider.PROVIDER_NAME);
```

Генерация симметричных ключей шифрования при помощи такого генератора `kg` будет осуществляться в соответствии с алгоритмом ГОСТ 28147-89 и с установленными в контрольной панели параметрами (параметрами по умолчанию). Если существует необходимость использования другого набора параметров (отличного от параметров по умолчанию), то следует установить требуемый набор параметров созданному генератору.

2.3.2. Определение параметров генерации симметричных ключей шифрования

После того, как генератор симметричных ключей был создан, может возникнуть необходимость установить некий набор параметров генерации симметричных ключей шифрования, отличный от параметров, установленных в контрольной панели. Операция изменения существующего набора параметров осуществляется при помощи метода *init(AlgorithmParameterSpec params)* класса KeyGenerator. Этому методу в качестве параметра может быть передан объект следующих двух классов:

- интерфейс `AlgIdInterface` (определяет набор параметров для генерации ключевой пары обмена, но может быть использован и для генерации симметричных ключей шифрования. В этом случае из передаваемого набора параметров в процессе генерации будут использованы лишь параметры шифрования);
- интерфейс `ParamsInterface` параметров алгоритма шифрования ГОСТ 28147-89 (см. «Работа с параметрами алгоритма шифрования ГОСТ 28147-89»).

В обоих случаях имеют значение лишь параметры шифрования (параметры алгоритма ГОСТ 28147-89), поскольку создаваемый генератором симметричный ключ может быть использован только для шифрования данных (операции создания ЭП, выработки ключей согласования и записи его на носитель запрещены).

Таким образом, изменение набора параметров генератора симметричных ключей шифрования производится следующим образом:

```
AlgIdInterface keyParams;     // интерфейс набора параметров

ParamsInterface cryptParams;  // интерфейс параметров шифрования

kg.init(keyParams);           // установка полного набора параметров

kg.init(cryptParams);         // установка параметров шифрования
```

Следует помнить о том, что изменение параметров генерации симметричных ключей имеет смысл только до выполнения непосредственно генерации.

В качестве параметров для инициализации генератора может использоваться любой из параметров шифрования, представленных провайдером: параметры по умолчанию, параметры шифрования 1, параметры шифрования 2, параметры шифрования 3, параметры Оскар 1.1, параметры Оскар 1.0, параметры РИК1, ТК26 2, ТК26 1, ТК26 3, ТК26 4, ТК26 5, ТК26 6, ТК26 Z.

При работе с ключами на алгоритме ГОСТ Р 34.10-2012 следует инициализировать генератор параметрами шифрования ТК26 Z.

2.3.3.Генерация симметричного ключа шифрования

Генерация симметричных ключей шифрования, соответствующих алгоритму ГОСТ 28147-89, осуществляется только после создания генератора и, если это необходимо, определения его параметров. Вызов метода *generateKey()* класса KeyGenerator возвращает ключ шифрования, соответствующий алгоритму ГОСТ 28147-89 и установленному набору параметров (или параметрам по умолчанию):

```
SecretKey key = kg.generateKey();
```

Пример генерации симметричного ключа шифрования при выполнении операции зашифрования данных см. `samples/samples_src.jar/userSamples/SessionEncrypt.java` (входит в комплект поставки программного обеспечения «КриптоПро JCP» версия 2.0).

3. Имитозащита данных в соответствии с алгоритмом ГОСТ 28147-89

Криптопровайдер «КриптоПро JCP» версия 2.0 (модули шифрования) осуществляет имитозащиту данных в соответствии с алгоритмом ГОСТ 28147-89, через стандартный интерфейс JCE при помощи класса Mac.

3.1. Создание объекта имитозащиты данных

Объект имитозащиты данных в соответствии с алгоритмом ГОСТ 28147-89 создается посредством вызова метода *getInstance()* класса Mac. Этот метод является статическим и возвращает ссылку на класс Mac, который обеспечивает выполнение требуемой операции.

Для создания объекта имитозащиты в соответствии с алгоритмом ГОСТ 28147-89 методу *getInstance()* необходимо в качестве параметра передать имя, идентифицирующее данный алгоритм ("GOST28147" или CryptoProvider.GOST_CIPHER_NAME). При вызове метода *getInstance()* совместно с определением требуемого алгоритма имитозащиты

данных осуществляется также определение требуемого типа криптопровайдера («КриптоПро JCP» версия 2.0 (модули шифрования)). Также стандартный интерфейс JCE позволяет в качестве параметра функции *getInstance()* класса Mac вместе с именем алгоритма указывать имя криптопровайдера, используемого для выполнения требуемой операции.

Таким образом, создание объекта имитозащиты данных осуществляется одним из следующих способов:

```
Mac mac = Mac.getInstance("GOST28147");

Mac mac = Mac.getInstance("GOST28147", "Crypto");

Mac mac = Mac.getInstance(CryptoProvider.GOST_CIPHER_NAME,
CryptoProvider.PROVIDER_NAME);
```

3.2. Инициализация объекта имитозащиты данных и определение его параметров

После того, как объект имитозащиты данных был создан, необходимо определить набор параметров алгоритма ГОСТ 28147-89, в соответствии с которыми будет осуществляться операция имитозащиты. Определение параметров алгоритма осуществляется во время инициализации операции имитозащиты данных методом *init()* класса Mac одним из следующих способов:

- при вызове функции *init(Key key)* параметры алгоритма ГОСТ 28147-89 определяются в соответствии с параметрами шифрования передаваемого функции ключа `key`;
- при вызове функции *init(Key key, AlgorithmParameterSpec params)* параметры алгоритма ГОСТ 28147-89 определяются в соответствии с передаваемыми параметрами `params`.

В обоих случаях передаваемый ключ `key` используется в процессе выработки имитовставки.

Существуют некоторые ограничения на параметры, передаваемые функции *init()*. В обоих способах вызова этой функции ключ `key` должен соответствовать типу SecretKey и удовлетворять алгоритму шифрования ГОСТ 28147-89 (такие ключи могут быть получены при помощи операции согласования ключей или при помощи генерации симметричных ключей шифрования). Необходимо также, чтобы параметры, передаваемые функции *init()* во втором способе ее вызова соответствовали интерфейсу `ParamsInterface` параметров алгоритма шифрования ГОСТ 28147-89 (см. «Работа с параметрами алгоритма шифрования ГОСТ 28147-89»). Таким образом, инициализация операции имитозащиты данных, во время которой происходит определение параметров имитовставки, осуществляется одним из следующих способов:

```
ParamsInterface cryptParams;    // интерфейс параметров шифрования

 SecretKey key;                    // обязательно ключ шифрования с алгоритмом
"GOST28147"
                                  // (соответствующий алгоритму ГОСТ 28147-89)
mac.init(key);

mac.init(key, cryptParams);
```

3.3. Копирование объекта имитозащиты данных

В некоторых случаях требуется создать копию уже существующего объекта имитозащиты данных, например, когда требуется выработать имитовставку как и части данных, так и всего исходного массива данных. В этом случае после того, как была обработана требуемая часть данных, необходимо сохранить (при помощи копирования) объект имитозащиты, и продолжить обработку оставшейся части (в результате чего будут обработаны все исходные данные). Уже после выполняется вычисление значения имитовставки для обоих объектов (исходного - соответствующего всем данным и скопированного - соответствующего части данных).

Для этих целей используется метод *clone()* класса Mac, который возвращает точную копию существующего объекта имитозащиты. Этот метод может быть вызван на любом этапе выполнения операции выработки имитовставки после того, как объект имитозащиты был проинициализирован и до того, как операция имитозащиты данных была завершена.

Использование метода клонирования объекта имитозащиты данных в следующем примере `samples/samples_src.jar/userSamples/CloneImita.java` (входит в комплект поставки программного обеспечения «КриптоПро JCP» версия 2.0).

3.4. Выработка имитовставки данных

После того, как объект имитозащиты был создан и проинициализирован, выработка имитовставки данных в соответствии с алгоритмом ГОСТ 28147-89 производится в два этапа: обработка данных и последующее вычисление значения имитовставки, завершающее операцию имитозащиты данных.

3.4.1.Обработка защищаемых данных

Обработка защищаемых данных осуществляется при помощи метода *update()* класса Mac. Этот метод осуществляет обработку защищаемых данных, представленных в виде байтового массива и подаваемых ему в качестве параметра. Существует 3 варианта обработки байтового массива данных при помощи этого метода:

1. Последовательная обработка каждого байта данных (при этом количество вызовов метода *update(byte b)* равно длине массива данных):

```
2.
3.      byte[] data;
4.      for(int i = 0; i < data.length; i++)
5.          mac.update(data[i]);
```

6. Блочная обработка данных (данные обрабатываются блоками определенной длины):

```
7.
8.      byte[] data;
9.      int BLOC_LEN = 1024;
10.
11.     //если длина исходных данных меньше длины блока
12.     if(data.length/BLOC_LEN == 0)
13.         mac.update(data);
14.     else {
15.         //цикл по блокам
16.         for (int i = 0; i < data.length/BLOC_LEN; i++) {
17.             byte[] bloc = new byte[BLOC_LEN];
18.             for(int j = 0; j < BLOC_LEN; j++)
19.                 bloc[j] = data[j + i * BLOC_LEN];
20.             mac.update(bloc);
21.         }
22.
23.         //обработка остатка
24.         byte[] endBloc = new byte[data.length % BLOC_LEN];
25.         for(int j = 0; j < data.length % BLOC_LEN; j++)
26.             endBloc[j] = data[j + data.length - data.length % BLOC_LEN];
27.         mac.update(endBloc);
28.     }
```

29. Обработка данных целиком:

```
30.
```

```
31.     byte[] data;
32.     mac.update(data);
```

Допускается комбинирование первого и второго варианта, обработка блоками различной длины, а также использование метода *update(byte[]data, int offset, int len)* - обработка массива данных со смещением.

3.4.2. Вычисление значения имитовставки

Вычисление значения имитовставки осуществляется при помощи метода *doFinal()* класса Mac. Существуют различные варианты вызова данного метода. Если обработке подверглись все защищаемые данные, то остается только получить значение имитовставки. Получить это значение можно двумя способами:

1. Вызов метода без параметров - *doFinal()*. В этом случае метод возвращает байтовый массив, содержащий значение имитовставки;
2. Вызов метода с параметрами - *doFinal(byte[] buf, int offset)*. В этом случае метод записывает значение имитовставки в передаваемый ему массив со смещением.

Если же часть данных осталось не обработанной, то следует предварительно обработать ее, а лишь потом получать значение имитовставки. Для этих целей используется функция *doFinal(byte[] buf)*, обрабатывающая переданные в массиве `buf` данные и возвращающая байтовый массив, содержащий значение имитовставки.

Пример создания и проверки имитовставки на ключах согласования сторон см. `samples/samples_src.jar/userSamples/CheckImita.java` (входит в комплект поставки программного обеспечения «КриптоПро JCP» версия 2.0).

3.5. Использование алгоритма HMAC

С помощью криптопровайдера «КриптоПро JCP» версия 2.0 можно осуществлять защиту данных в соответствии с алгоритмом HMAC_GOSTR3411, HMAC_GOSTR3411_2012_256 (OID: 1.2.643.7.1.1.4.1) или HMAC_GOSTR3411_2012_512 (OID: 1.2.643.7.1.1.4.2). Стандарт HMAC_GOSTR3411 описан в документе rfc4357, который базируется на стандарте rfc2104. OID "1.2.643.2.2.10" для алгоритма определен в rfc4490

Работать с реализацией алгоритма можно через стандартный интерфейс JCE при помощи класса Mac Использование HMAC_GOSTR3411, HMAC_GOSTR3411_2012_256 или HMAC_GOSTR3411_2012_512 аналогично работе с имитозащитой по алгоритму ГОСТ 28147-89, только при создании объекта в качестве имени надо указывать "HMAC_GOSTR3411". Примеры создания объекта:

```
Mac mac = Mac.getInstance("HMAC_GOSTR3411");
```

```
Mac mac = Mac.getInstance("HMAC_GOSTR3411", "Crypto");
```

```
Mac mac = Mac.getInstance(ru.CryptoPro.Crypto.Cipher.GostHMAC.STR_NAME,
CryptoProvider.PROVIDER_NAME);
```

```
Mac mac = Mac.getInstance("1.2.643.2.2.10", "Crypto");
```

или

```
Mac mac = Mac.getInstance("HMAC_GOSTR3411_2012_256");
```

```
Mac mac = Mac.getInstance("HMAC_GOSTR3411_2012_256", "Crypto");
```

```
Mac mac =
Mac.getInstance(ru.CryptoPro.Crypto.Cipher.GostHMAC2012_256.STR_NAME,
CryptoProvider.PROVIDER_NAME);

Mac mac = Mac.getInstance("1.2.643.7.1.1.4.1", "Crypto");
```

или

```
Mac mac = Mac.getInstance("HMAC_GOSTR3411_2012_512");

Mac mac = Mac.getInstance("HMAC_GOSTR3411_2012_512", "Crypto");

Mac mac =
Mac.getInstance(ru.CryptoPro.Crypto.Cipher.GostHMAC2012_512.STR_NAME,
CryptoProvider.PROVIDER_NAME);

Mac mac = Mac.getInstance("1.2.643.7.1.1.4.2", "Crypto");
```

3.5.1. Использование 512-битных ключей в алгоритмах HMAC

В функциях HMAC, основанных на хэш-функциях ГОСТ Р 34.11-2012 (256 бит) и ГОСТ Р 34.11-2012 (512 бит), могут быть использованы 512-битные ключи. Эти ключи представляются в виде объектов класса Symmetric512Key.

Ключи Symmetric512Key могут быть получены как путем выработки с помощью алгоритма ВКО (см. соответствующий раздел), так и с помощью объекта генератора Symmetric512KeyGenerator.

```
KeyGenerator kg = KeyGenerator.getInstance("SYMMETRIC512");

KeyGenerator kg = KeyGenerator.getInstance("SYMMETRIC512", "Crypto");

KeyGenerator kg = KeyGenerator.getInstance(CryptoProvider.SYMMETRIC_512,
CryptoProvider.PROVIDER_NAME);
```

Непосредственно выработка ключа производится так:

```
kg.init(null);

SecretKey key = kg.generateKey();
```

После этого ключ можно использовать в HMAC:

```
Mac mac;

Symmetric512Key key;

mac.init(key);
```

4. Шифрование данных и ключей в соответствии с алгоритмом ГОСТ 28147-89

Криптопровайдер «КриптоПро JCP» версия 2.0 (модули шифрования) осуществляет шифрование данных и ключей в соответствии с алгоритмом ГОСТ 28147-89, через стандартный интерфейс JCE при помощи класса Cipher.

4.1. Создание объекта шифрования данных и ключей (шифратора)

Объект шифрования данных и ключей (далее *шифратор*) в соответствии с алгоритмом ГОСТ 28147-89 создается посредством вызова метода *getInstance()* класса Cipher. Этот метод является статическим и возвращает ссылку на класс Cipher, который обеспечивает выполнение требуемой операции.

Следуя интерфейсу класса Cipher, методу *getInstance()* в качестве параметра следует передавать строковое представление алгоритма шифратора вида *"algorithm/mode/padding"* либо *"algorithm"*. Ниже описаны допустимые варианты значений каждого из элементов такого строкового представления.

1. в качестве элемента *"algorithm"* указывается собственно алгоритм шифрования. Алгоритм ГОСТ 28147-89 идентифицируется именем "GOST28147" или CryptoProvider.GOST_CIPHER_NAME.
2. в качестве элемента *"mode"* указывается режим шифрования. В криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования) допустимы следующие режимы:
   - "ECB" - режим простой замены (используется при шифровании данных);
   - "CBC" - режим простой замены с зацеплением (используется при шифровании данных);
   - "CNT" - режим гаммирования (используется при шифровании данных);
   - "CFB" - режим гаммирования с обратной связью (используется при шифровании данных);
   - "SIMPLE_EXPORT" - режим шифрования симметричного ключа без усложнения (используется при шифровании ключей);
   - "PRO_EXPORT" - режим шифрования симметричного ключа с усложнением (используется при шифровании ключей);
   - "PRO12_EXPORT" - режим шифрования симметричного ключа с усложнением (используется при шифровании ключей).

Если элемент *"mode"* отсутствует (т.е. в качестве строкового представления алгоритма шифратора указан лишь алгоритм шифрования), то при осуществлении последующей операции шифрования будет использован режим по умолчанию: для операций шифрования данных (см. ниже) используется режим "CFB", для операций шифрования ключей - режим "SIMPLE_EXPORT".

3. в качестве элемента *"padding"* указывается алгоритм заполнения последнего неполного блока данных при выполнении операции блочного шифрования. В криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования) допустимы следующие алгоритмы:
   - "NoPadding" - заполнение последнего блока не используется (использование допускается только в режимах CNT и CFB);
   - "PKCS5_PADDING" - заполнение последнего блока осуществляется по алгоритму PKCS5;
   - "PKCS5Padding" - другой допустимый вариант написания алгоритма PKCS5;
   - "ISO10126Padding" - заполнение последнего блока осуществляется по алгоритму ISO 10126;
   - "ANSIX923Padding" - заполнение последнего блока осуществляется по алгоритму ANSI X.923;
   - "ZeroPadding" - заполнение последнего блока нулями, при кратности исходного открытого текста размеру блока шифра дополнительный блок не добавляется, паддинг не является однозначным;
   - "RandomPadding" - заполнение последнего блока случайными байтами, при кратности исходного открытого текста размеру блока шифра дополнительный блок не добавляется, паддинг не является однозначным.

Если элемент *"padding"* отсутствует (т.е. в качестве строкового представления алгоритма шифратора указан лишь алгоритм шифрования), то при осуществлении последующей операции шифрования в режимах "ECB" и "CBC" будет использован алгоритм "PKCS5_PADDING", в режимах "CNT" или "CFB" – "NoPadding".

Следует помнить, что установление отличного от "NoPadding" алгоритма заполнения имеет смысл только при работе в режимах "ECB" и "CBC". Если используется режим шифрования "CNT" или "CFB", то установленный режим заполнения будет игнорироваться. Выполнение операций зашифрования и расшифрования в режимах "ECB" и "CBC" с использованием режима заполнения "NoPadding" не допускается.

При вызове метода *getInstance()* со строковым представлением алгоритма, состоящем из перечисленных выше допустимых значений имени алгоритма, режима шифрования и алгоритма заполнения последнего неполного блока, помимо определения в

соответствии с этим строковым представлением алгоритма работы шифратора осуществляется также определение требуемого типа криптопровайдера («КриптоПро JCP» версия 2.0 (модули шифрования)). Также стандартный интерфейс JCE позволяет в качестве параметра функции *getInstance()* класса Cipher вместе с именем алгоритма указывать имя криптопровайдера, используемого для выполнения требуемой операции. Таким образом, создание, например, объекта шифрования данных в соответствии с алгоритмом ГОСТ 28147-89 в режиме простой замены с зацеплением с алгоритмом PKCS5 заполнения последнего неполного блока осуществляется одним из следующих способов:

```
Cipher cipher = Cipher.getInstance("GOST28147/CBC/PKCS5_PADDING");
```

```
Cipher cipher = Cipher.getInstance("GOST28147/CBC/PKCS5_PADDING", "Crypto");
```

4.2. Инициализация шифратора и определение его параметров

После того, как шифратор был создан, необходимо определить собственно метод шифрования и набор параметров, в соответствии с которым будет осуществляться операция шифрования. Инициализация шифратора осуществляется посредством вызова метода *init()* класса Cipher. Вызов данного метода может быть осуществлен двумя способами:

- *init(int opmode, Key key)*;
- *init(int opmode, Key key, AlgorithmParameterSpec params)*.

Ниже описываются особенности каждого из этих способов для криптопровайдера «КриптоПро JCP» версия 2.0 (модули шифрования).

- для обоих случаев вызова функции *init()* в качестве параметра `opmode` передается число, определяющее собственно метод шифрования. Криптопровайдер «КриптоПро JCP» версия 2.0 (модули шифрования) поддерживает четыре метода шифрования, определяемых следующим образом:
  - метод зашифрования данных (определяется константой `ENCRYPT_MODE` класса Cipher);
  - метод расшифрования данных (определяется константой `DECRYPT_MODE` класса Cipher);
  - метод зашифрования ключа (определяется константой `WRAP_MODE` класса Cipher);
  - метод расшифрования ключа (определяется константой `UNWRAP_MODE` класса Cipher);
- для обоих случаев в качестве параметра `key` передается ключ, на котором будет производится указанная операция шифрования. Требуется, чтобы этот ключ являлся объектом типа SecretKey и удовлетворял алгоритму шифрования ГОСТ 28147-89 (такие ключи могут быть получены при помощи операции согласования ключей или при помощи генерации симметричных ключей шифрования).
- поскольку в первом способе вызова функции *init(int opmode, Key key)* никакие дополнительные параметры не передаются, то такой вызов функции обладает следующими особенностями:
  - параметры алгоритма шифрования (узел замены) определяются в соответствии с параметрами шифрования передаваемого ключа `key`;
  - при инициализации операции зашифрования данных (`ENCRYPT_MODE`) в режиме, требующем вектора инициализации (такими режимами являются "CBC", "CFB" и "CNT") вектор инициализации будет выработан случайным образом;

- при инициализации операции расшифрования данных (`DECRYPT_MODE`) в режиме, требующем вектора инициализации (такими режимами являются "CBC", "CFB" и "CNT") будет выдано исключение;

- при инициализации операции шифрования ключа (`WRAP_MODE` или `UNWRAP_MODE`) ключом `key`, полученным при помощи операции согласования ключей, необходимый для дальнейшего выполнения операции шифрования вектор инициализации будет пронаследован от стартового вектора, который участвовал в процессе создания ключа согласования `key`;

- при инициализации операции зашифрования ключа (`WRAP_MODE`) ключом `key`, полученным при помощи генерации симметричных ключей шифрования, необходимый для дальнейшего выполнения операции шифрования вектор инициализации будет выработан случайным образом;

- при инициализации операции расшифрования ключа (`UNWRAP_MODE`) ключом `key`, полученным при помощи генерации симметричных ключей шифрования, будет выдано исключение.

- во втором способе вызова функции *init(int opmode, Key key, AlgorithmParameterSpec params)* передаются параметры `params`, определяющие работу шифратора. В качестве таких параметров в криптопровайдере «КриптоПро JCP» версия 2.0 (модули шифрования) допускается передавать объекты следующих типов:

  - интерфейс `ParamsInterface` параметров алгоритма шифрования ГОСТ 28147-89 (описание работы с этим интерфейсом приводится в соответствующем разделе документации по криптопровайдеру «КриптоПро JCP» версия 2.0). Таким образом, объект типа `ParamsInterface`, передаваемый в качестве параметров `params` определяет параметры шифрования (узел замены)

  - IvParameterSpec - стандартный класс JCE. Объект такого класса, переданный в качестве параметров `params` определяет вектор инициализации.

  - `GostCipherSpec` - этот класс представляет собой набор параметров алгоритма ГОСТ 28147-89, а также вектор инициализации шифратора, и является реализацией стандартного класса AlgorithmParameterSpec. Создание объекта класса `GostCipherSpec` в соответствии с требуемыми параметрами (узлом) шифрования и вектором инициализации производится следующим образом:

```
ParamsInterface cryptParams; // интерфейс параметров шифрования
byte[] iv;                   //вектор инициализации
GostCipherSpec spec = new GostCipherSpec(iv, cryptParams);
```

Таким образом, объект типа GostCipherSpec, передаваемый в качестве параметров params определяет параметры шифрования (узел замены) и вектор инициализации.

- ввиду поддерживаемых типов объектов, передаваемых в качестве параметров `params` для второго способа вызова функции *init(int opmode, Key key, AlgorithmParameterSpec params)* возникают следующие особенностями такого вызова функции:

  - если параметры алгоритма шифрования (узел замены) заданы (в случае передачи в качестве параметров `params` объекта интерфейса параметров шифрования `ParamsInterface` или класса `GostCipherSpec`), то они и будут использованы в процессе выполнения операции шифрования. В противном случае, параметры шифрования определяются в соответствии с параметрами шифрования передаваемого ключа `key`;

  - если вектор инициализации задан (в случае передачи в качестве параметров `params` объекта класса `IvParameterSpec` или класса `GostCipherSpec`), то он и

будет использован в процессе выполнения операции шифрования. В противном случае:

- при инициализации операции зашифрования данных (`ENCRYPT_MODE`) в режиме, требующем вектора инициализации (такими режимами являются "CBC", "CFB" и "CNT") вектор инициализации будет сгенерирован случайным образом;
- при инициализации операции расшифрования данных (`DECRYPT_MODE`) в режиме, требующем вектора инициализации (такими режимами являются "CBC", "CFB" и "CNT") будет выдано исключение;
- при инициализации операции шифрования ключа (`WRAP_MODE` или `UNWRAP_MODE`) ключом `key`, полученным при помощи операции согласования ключей, необходимый для дальнейшего выполнения операции шифрования вектор инициализации будет пронаследован от стартового вектора, который участвовал в процессе создания ключа согласования `key`;
- при инициализации операции зашифрования ключа (`WRAP_MODE`) ключом `key`, полученным при помощи генерации симметричных ключей шифрования, необходимый для дальнейшего выполнения операции шифрования вектор инициализации будет выработан случайным образом;
- при инициализации операции расшифрования ключа (`UNWRAP_MODE`) ключом `key`, полученным при помощи генерации симметричных ключей шифрования, будет выдано исключение.

В обоих случаях необходимо следить за согласованностью параметров шифрования (узла шифрования), а также вектора инициализации при выполнении соответствующих друг другу операций зашифрования и расшифрования. Так, например, если инициализация операции зашифровании данных производилась без передачи вектора инициализации, то созданный в процессе выполнения этой операции вектор инициализации следует передать вместе с зашифрованным текстом для корректного выполнения операции расшифрования. Получить вектор инициализации можно при помощи метода *getIV()*, возвращающего его в виде байтового массива. Аналогичные действия следует производить в случае зашифрования ключа на симметричном ключе шифрования, полученным при помощи генерации. Необходимо помнить, что вызов этой функции следует производить только после того, как шифратор был проинициализирован, и до того как операция зашифрования была завершена.

Таким образом, инициализация, соответствующих друг другу операций зашифрования и расшифрования данных для алгоритма ГОСТ 28147-89 может быть осуществлена следующим образом:

```
SecretKey key;     // обязательно ключ шифрования с алгоритмом "GOST28147"
                   // (соответствующий алгоритму ГОСТ 28147-89)
                   // считаем, что известен обоим сторонам

/*Зашифрование*/

// инициализация операции зашифрования. Вектор будет сгенерирован
// в процессе выполнения операции
cipher.init(Cipher.ENCRYPT_MODE, key);

// получение сгенерированного вектора инициализации
byte[] iv = cipher.getIV();
```

```
/*Расшифрование*/


// создание параметров, содержащих требуемый вектор
IvParameterSpec spec = new IvParameterSpec(iv);


// инициализация операции расшифрования с тем же
// ключом и вектором
cipher.init(Cipher.DECRYPT_MODE, key, spec);
```

4.3. Зашифрование и расшифрование данных

После того, как шифратор был создан и проинициализирован методом шифрования данных (`ENCRYPT_MODE` или `DECRYPT_MODE`), операция зашифрования (расшифрования) данных производится в два этапа: обработка данных и последующее завершение операции.

4.3.1.Последовательное зашифрование (расшифрование) данных

Последовательное зашифрование (расшифрование) данных осуществляется при помощи метода *update()* класса Cipher. Этот метод выполняет последовательное зашифрование (расшифрование) данных, представленных в виде байтового массива и подаваемых ему в качестве параметра. Существует четыре способа вызова метода:

- *update(byte[] input)* - зашифрование (расшифрования) всего содержимого массива `input`. Результат шифрования выдается в качестве байтового массива;
- *update(byte[] input, int inputOffset, int inputLen)* - зашифрование (расшифрования) содержимого массива `input`, начиная с позиции `inputOffset` в количестве `inputLen` байт. Результат шифрования выдается в качестве байтового массива;
- *update(byte[] input, int inputOffset, int inputLen, byte[] output)* - зашифрование (расшифрования) содержимого массива `input`, начиная с позиции `inputOffset` в количестве `inputLen` байт. Результат шифрования записывается в массив `output`. Такой вызов функции возвращает количество записанных в выходной буфер байт;
- *update(byte[] input, int inputOffset, int inputLen, byte[] output, int outputOffset)* - зашифрование (расшифрования) содержимого массива `input`, начиная с позиции `inputOffset` в количестве `inputLen` байт. Результат шифрования записывается в массив `output`, начиная с позиции `outputOffset`. Такой вызов функции возвращает количество записанных в выходной буфер байт;

Допускается комбинирование этих методов, причем подаваемые на каждый вызов функции *update()* массивы зашифровываемых (расшифровываемых) данных могут быть разной длины.

4.3.2.Завершение операции зашифрования (расшифрования)

Завершение операции зашифрования (расшифрования) данных осуществляется при помощи метода *doFinal()* класса Cipher. Даже если все входные данные были поданы на последовательное шифрование, то это не гарантирует того, что результат шифрования был выдан полностью. Такое, например, возможно при блочном зашифровании данных, длина которых не кратна длине блока. Ввиду этого операцию шифрования следует завершить (для получения последней части результата шифрования, либо же для обработки и получения конечного результата еще не обработанных входных данных).

Если в процессе последовательного шифрования все входные данные были поданы на зашифрование (расшифрование), то завершить операцию шифрования можно двумя способами:

- вызов метода без параметров - *doFinal()*. В этом случае метод возвращает байтовый массив, содержащий последнюю часть зашифрованных (расшифрованных) данных;

- вызов метода с параметрами - *doFinal(byte[] output, int outputOffset)*. В этом случае метод записывает последнюю часть зашифрованных (расшифрованных) данных в передаваемый ему массив со смещением.

Если же не все входные данные были поданы на последовательное шифрование, то следует обработать оставшиеся данные, и лишь потом завершать операцию шифрования для получения конечного результата. Для этих целей может быть использовано несколько вариантов вызова функции *doFinal*:

- *doFinal(byte[] input)* - обрабатывает последнюю часть входных данных, содержащуюся в массиве `input` и выдает окончательный результат шифрования в виде байтового массива;
- *doFinal(byte[] input, int inputOffset, int inputLen)* - обрабатывает последнюю часть входных данных, содержащуюся в массиве `input`, начиная с позиции `inputOffset` в количестве `inputLen` и выдает окончательный результат шифрования в виде байтового массива;
- *doFinal(byte[] input, int inputOffset, int inputLen, byte[] output)* - обрабатывает последнюю часть входных данных, содержащуюся в массиве `input`, начиная с позиции `inputOffset` в количестве `inputLen` и записывает окончательный результат шифрования в массив `output`. При этом метод возвращает количество записанных в выходной буфер байт;
- *doFinal(byte[] input, int inputOffset, int inputLen, byte[] output, int outputOffset)* - обрабатывает последнюю часть входных данных, содержащуюся в массиве `input`, начиная с позиции `inputOffset` в количестве `inputLen` и записывает окончательный результат шифрования в массив `output`, начиная с позиции `outputOffset`. При этом метод возвращает количество записанных в выходной буфер байт.

Примеры зашифрования и расшифрования данных входят в комплект поставки программного обеспечения «КриптоПро JCP» версия 2.0: `samples/samples_src.jar/userSamples/SessionEncrypt.java` - пример шифрования данных на симметричном ключе (соответственно, с зашифрованием этого ключа для передачи его противоположной стороне), `samples/samples_src.jar/userSamples/ClassicEncrypt.java` - пример шифрования данных по классической схеме (на ключах согласования противоположных сторон).

4.4. Зашифрование и расшифрование ключей

После того, как шифратор был создан и проинициализирован методом шифрования ключа (`WRAP_MODE` или `UNWRAP_MODE`), выполняется собственно операция зашифрования или расшифрования ключа.

4.4.1. Зашифрование ключа

Операция зашифрования ключа выполняется методом *wrap(Key key)* класса Cipher. В качестве параметра данному методу подается ключ `key`, который подлежит зашифрованию. Зашифрование переданного ключа производится на ключе, которым был проинициализирован шифратор. Криптопровайдер «КриптоПро JCP» версия 2.0 (модули шифрования) допускает зашифрование только следующих типов ключей:

- закрытых ключей обмена или ключей электронной подписи, соответствующих алгоритмам обмена Диффи-Хелмана и ЭЦП ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012 (такие ключи могут быть прочитаны из контейнера, либо могут быть получены посредством генерации ключевой пары в соответствии с данными алгоритмами);

- симметричных ключей шифрования, соответствующих алгоритму ГОСТ 28147-89 (такие ключи могут быть получены при генерации симметричных ключей).

Зашифрованный ключ возвращается в виде байтового массива.

4.4.2. Расшифрование ключа

Операция расшифрования ключа выполняется методом *unwrap(byte[] wrappedKey, String wrappedKeyAlgorithm, int wrappedKeyType)* класса Cipher. Этот метод возвращает расшифрованный ключ. В качестве параметра данному методу передается зашифрованный ключ `wrappedKey`, представленный в виде байтового массива. Параметр `wrappedKeyAlgorithm` в криптопровайдере «КриптоПро JCP» версия 2.0 (модули

шифрования) не используется. Расшифрование ключа производится на ключе, которым был проинициализирован шифратор. Криптопровайдер «КриптоПро JCP» версия 2.0 (модули шифрования) допускает расшифрование только симметричных ключей шифрования (соответствующих типу SecretKey и удовлетворяющих алгоритму шифрования ГОСТ 28147-89). Ввиду этого в качестве параметра `wrappedKeyType` допускает использование только типа ключа, соответствующего классу SecretKey. Тип такого ключа определяется константой `SECRET_KEY` класса Cipher.

Пример зашифрования и расшифрования симметричного ключа, на котором осуществляется зашифрования и расшифрование данных см. `samples/samples_src.jar/userSamples/SessionEncrypt.java` (входит в комплект поставки программного обеспечения «КриптоПро JCP» версия 2.0).

Примеры зашифрования/расшифрования сессионного ключа и текста на нем, совместимые с примерами зашифрования/расшифрования с использованием КриптоПро CSP см. `samples/samples_src.jar/Crypt_samples` (входит в комплект поставки программного обеспечения «КриптоПро JCP» версия 2.0).

5. Шифрование XML-документов

Использование криптопровайдера «КриптоПро JCP» версия 2.0 совместно с библиотекой Apache XML Security позволяет шифровать XML-документы по алгоритму ГОСТ 28147-89. Описание совместной работы «КриптоПро JCP» версия 2.0 с библиотекой XML Security и способы регистрации ГОСТ алгоритмов приводятся в документе «Руководство программиста» «КриптоПро JCP» версия 2.0.

Константы, задающие имена алгоритмов, определены в файле ru.CryptoPro.JCPxml.Consts

```
/**
 * алгоритм шифрования данных
 */
public static final String URI_GOST_CIPHER =
        "urn:ietf:params:xml:ns:cpxmlsec:algorithms:gost28147";
/**
 * URI алгоритма шифрования ключа
 */
public static final String URI_GOST_TRANSPORT =
        "urn:ietf:params:xml:ns:cpxmlsec:algorithms:transport-gost2001";
```

Пример CryptXML шифрования XML-документа входит в комплект поставки программного обеспечения «КриптоПро JCP» версия 2.0 `samples/samples_src.jar/xmlSign/`.