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 JTLS

Версия 2.0

Руководство программиста

ЖТЯИ.00091-01 33 03
Листов 23

2016 г.

Оглавление

1. Введение

КриптоПро JTLS является программным комплексом защиты информации, разработанным на основе «КриптоПро JCP» версия 2.0 и реализующим протоколы SSL и TLS в соответствии с российскими криптографическими алгоритмами.

Основные функции, реализуемые «КриптоПро JTLS» версия 2.0:

- Две схемы аутентификации с использованием обмена ключей по алгоритму Диффи-Хэллмана и хэширования в соответствии с ГОСТ Р 34.11-94 или ГОСТ Р 34.11-2012.
  - односторонняя - анонимный клиент, аутентифицируемый сервер;
  - двухсторонняя - аутентифицируемые клиент и сервер.

  В случае аутентификации клиента на ключе подписи применяются алгоритмы выработки электронной подписи в соответствии с ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012 и проверки в соответствии с ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012.
- шифрование соединения в соответствии с ГОСТ 28147-89;
- имитозащита передаваемых данных в соответствии с ГОСТ 28147-89.

2. Установка «КриптоПро JTLS» версия 2.0

Предварительно установить «КриптоПро JCP» версия 2.0 (включая модули шифрования). Далее установку можно выполнить двумя путями:

- Выполнить команду "`<JRE>/bin/java -jar cpSSL.jar`" и затем ввести серийный номер через контрольную панель
- Выполнить установку с вводом серийного номера

```
<JRE>/bin/java -cp cpSSL.jar ru.CryptoPro.ssl.JTLSInstall -install
-verbose -sslserial XXXXX-XXXXX-XXXXX-XXXXX-XXXXX -sslcompany "My Company"
```

Вместо `-sslcompany "My Company"` возможно использование ключа `-sslcombase "company_name_in_base64"`, которому передается название компании в BASE64.

Использование «КриптоПро JTLS» версия 2.0 возможно только на Java 1.6 и выше. Для преодоления экспортных ограничений на стойкую криптографию см. «Особенности подключения «КриптоПро JTLS» версия 2.0».

Установка модуля так же может быть выполнена с помощью графического (setup.exe, setup_gui.sh <JRE>) или консольного (setup_console.bat <JRE>, setup_console.sh <JRE>) инсталляторов, как отдельного компонента «КриптоПро JCP» версия 2.0 (см. «Руководство администратора безопасности»).

3. Работа через внешний интерфейс JSSE

ПКЗИ «КриптоПро TLS» версия 2.0 реализует стандартный интерфейс Java Secure Socket Extension (JSSE) v.1.6 и обеспечивает выполнение защищенной передачи данных по протоколам SSL и TLS в соответствии с российскими криптографическими алгоритмами через стандартный интерфейс JSSE.

3.1. Особенности подключения «КриптоПро JTLS» версия 2.0

Из-за того, что обычно в JSSE, помимо самого интерфейса, включаются и некоторые его реализации (примером такой реализации на SUNовской виртуальной машине является провайдер `com.sun.net.ssl.internal.ssl.Provider`), существуют некоторые особенности использования ПКЗИ «КриптоПро JTLS» версия 2.0.

Возможна ситуация, когда установленная JRE имеет экспортные ограничения. США запрещает экспорт "сильной" криптографии и «КриптоПро JCP» версия 2.0 с длиной ключа 256 бит попадает под это ограничение. Ограничения устанавливаются файлами `local_policy.jar` и `US_export_policy.jar` в каталоге `<JRE>/jre/lib/security`. Для снятия экспортных ограничений необходимо скачать файл `jce_policy.zip` с политиками со страницы http://www.oracle.com/technetwork/java/javase/downloads/index.html, выбирая "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files" версии 6 или 7. Для отладки же можно просто скопировать `US_export_policy.jar` в `local_policy.jar` (оба файла должны присутствовать).

3.2. Использование «КриптоПро JTLS» версия 2.0 через внешний интерфейс JSSE

После того, как «КриптоПро JTLS» версия 2.0 подключен, дальнейшая работа с протоколами SSL и TLS в соответствии с российскими криптографическими алгоритмами осуществляется через стандартный интерфейс JSSE, например, при помощи метода `getDefault()` классов SSLServerSocketFactory и SSLSocketFactory.

- Если данная сторона является сервером, то установление защищенного по протоколам SSL и TLS соединения осуществляется при помощи функциональности класса SSLServerSocket. Для получения объекта такого класса, имеющего возможность осуществлять защищенный обмен данными в соответствии с российскими криптографическими алгоритмами, необходимо сделать следующее:

```
// порт, по которому данный сервер устанавливает соединение
int port;

SSLServerSocketFactory sslSrvFact =
(SSLServerSocketFactory) SSLServerSocketFactory
.getDefault();

SSLServerSocket ss = (SSLServerSocket)sslSrvFact.createServerSocket(port);
```

Защищенное соединение в общем случае может быть осуществлено и при помощи класса ServerSocket (объекты именно этого класса возвращаются методом createServerSocket(port)). Однако, такой класс имеет меньшую функциональность, чем его расширение - класс SSLServerSocket. Выбор того, каким классом пользоваться, базируется на необходимости установления тех или иных атрибутов процесса обмена.

- Если данная сторона является клиентом, то установление защищенного по протоколам SSL и TLS соединения осуществляется при помощи функциональности класса SSLSocket. Для получения объекта такого класса, имеющего возможность осуществлять защищенный обмен данными в соответствии с российскими криптографическими алгоритмами, необходимо сделать следующее:

```
// порт сервера, по которому устанавливается соединение
int port;

// имя хоста сервера, по которому устанавливается соединение
String host;

SSLSocketFactory sslFact =
 (SSLSocketFactory) SSLSocketFactory.getDefault();
SSLSocket soc = (SSLSocket)sslFact.createSocket(host, port);
```

Защищенное соединение в общем случае может быть осуществлено и при помощи класса Socket (объекты именно этого класса возвращаются методом createSocket(host, port)). Однако, такой класс имеет меньшую функциональность, чем его расширение - класс SSLSocket. Выбор того, каким классом пользоваться, базируется на необходимости установления тех или иных атрибутов процесса обмена.

Другой способ создать защищенный SSL контекст:

```
KeyStore trustStore = KeyStore.getInstance(JCP.CERT_STORE_NAME);
trustStore.load(new FileInputStream("path_to_trust_store"),
  "trust_store_password".toCharArray()); // хранилище корневых сертификатов

// Если контекст для сервера или для клиента с аутентификацией
if (isServer || clientNeedsAuth) {

  KeyManagerFactory kmf = KeyManagerFactory.getInstance("GostX509");
  KeyStore keyStore = KeyStore.getInstance(JCP.HD_STORE _NAME);

  keyStore.load(null, null);
  kmf.init(keyStore, "key_store_password".toCharArray()); // Пароль к контейнеру
  сервера или клиента

} // if

TrustManagerFactory tmf = TrustManagerFactory.getInstance("GostX509");
tmf.init(trustStore);

SSLContext sslCtx = SSLContext.getInstance("GostTLS"); // Защищенный контекст
sslCtx.init(kmf != null ? kmf.getKeyManagers() : null,
  tmf.getTrustManagers(), null);
```

Из SSLContext далее можно получить SSLSocketFactory и создавать сокеты.

Примеры создания защищенного соединения между клиентом и сервером по протоколу TLS приводятся в Samples/JTLS_samples, входящих в пакет поставки «КриптоПро JCP» версия 2.0. Пакет ComLine модуля Samples содержит примеры сервера и клиента, запускаемые из командной строки.

4. Управление ключами и сертификатами

В ПКЗИ «КриптоПро JTLS» версия 2.0 в процессе установления защищенного соединения по протоколам SSL и TLS используются следующие три типа объектов:

- закрытый ключ обмена;
- сертификат (или цепочка сертификатов) открытого ключа, соответствующего закрытому ключу;
- доверенный сертификат для проверки сертификата (или цепочки сертификатов) противоположной стороны.

В зависимости от роли данной стороны (клиент или сервер), а также от типа аутентификации (односторонняя или двусторонняя) существуют особенности использования данных объектов.

4.1. Управление ключами и сертификатами сервером

4.1.1. Общие положения

В ПКЗИ «КриптоПро JTLS» версия 2.0 аутентификация может быть односторонней или двусторонней (т.е. сервер ВСЕГДА отправляет свой сертификат аутентификации клиенту). Поэтому, если данная сторона является сервером, то необходимо, чтобы у нее существовали закрытый ключ и соответствующий ему сертификат (цепочка сертификатов) аутентификации, в соответствии с которыми и будет устанавливаться защищенное соединение. Причем, допустимы к использованию только следующие пары (закрытый ключ обмена и соответствующий ему сертификат аутентификации):

- закрытый ключ соответствует алгоритмам обмена Диффи-Хэллмана и подписи ГОСТ 34.10-2001;
- сертификат имеет следующие расширения: расширение "Использования ключа" включает в себя "Шифрование ключей" или "Согласование ключей"; расширение "Улучшенный ключ" имеет значение "Проверка подлинности сервера".

4.1.2. Действия перед началом обмена с клиентом

Таким образом, перед началом осуществления соединения с клиентом, такие закрытый ключ и соответствующий ему сертификат (цепочка сертфикат) необходимо создать и затем положить в ключевое хранилище, из которого они и будут прочитаны в процессе обмена. Методы выполнения этих операций подробно описаны в руководстве программиста «КриптоПро JCP» версия 2.0, а также в руководстве программиста (модули шифрования). Ниже приводятся примеры таких методов.

Создание закрытого ключа обмена и соответствующего ему открытого ключа осуществляется при помощи методов стандартного класса `KeyPairGenerator`, проинициализированного именем `"GOST3410DHEL"` (это имя соответствует алгоритмам обмена Диффи-Хэллмана и подписи ГОСТ Р 34.10-2001):

```
KeyPairGenerator kg = KeyPairGenerator.getInstance("GOST3410DHEL");
KeyPair pair = kg.generateKeyPair();
// закрытый ключ обмена
PrivateKey privKey = pair.getPrivate();
// соответствующий ему открытый ключ
PublicKey pubKey = pair.getPublic();
```

Создание сертификата аутентификации при помощи методов класса `GostCertificateRequest`, представляющего собой дополнительную возможность работы с сертификатами, реализованную на базе СКЗИ «КриптоПро JCP» версия 2.0:

```
String keyAlg = "GOST3410DHEL";
String certName = "CN=newCert, O=CryptoPro, C=RU";
```

```
        String httpAddress = "http://www.cryptopro.ru/certsrv/";

        // создание запроса на сертификат аутентификации сервера
        GostCertificateRequest request = new GostCertificateRequest();
        request.setKeyUsage(GostCertificateRequest.CRYPT_DEFAULT);
        request.addExtKeyUsage(GostCertificateRequest.INTS_PKIX_CLIENT_AUTH);
        request.addExtKeyUsage(GostCertificateRequest.INTS_PKIX_SERVER_AUTH);
        request.setPublicKeyInfo(pubKey);
        request.setSubjectInfo(certName);
        request.encodeAndSign(privKey);

        // отправка запроса центру сертификации и получение от центра
        // сертификата в DER-кодировке
        byte[] encoded = request.getEncodedCert(httpAddress);

        // генерация X509-сертификата из закодированного представления сертификата
        CertificateFactory cf = CertificateFactory.getInstance("X509");
        java.security.cert.Certificate cert = cf.generateCertificate(new
ByteArrayInputStream(encoded));
```

Запись созданного закрытого ключа и цепочки сертификатов, состоящей из сертификата аутентификации и корневого сертификата центра сертификации, на жесткий диск:

```
        String password = "password";
        String alias = "newKey";
        String certPath = "C:\\certificate.cer"; // файл, в который был предварительно
                                                 // сохранен корневой сертификат
центра

        CertificateFactory cf = CertificateFactory.getInstance("X509");
        FileInputStream fis = new FileInputStream(certPath);
        java.security.cert.Certificate certRoot = cf.generateCertificate(new
BufferedInputStream(fis));

        java.security.cert.Certificate[] certs = new
java.security.cert.Certificate[2];
        certs[0] = certRoot;
        certs[1] = cert;

        KeyStore hdImageStore = KeyStore.getInstance("HDImageStore");
        hdImageStore.load(null, null);
        hdImageStore.setKeyEntry(alias, privKey, password.toCharArray(), certs);
        hdImageStore.store(null, null);
```

В данных примерах приводилось использование тестового центра сертификации КриптоПро.

Также можно воспользоваться готовыми классами пакета ComLine из модуля Samples, входящего в состав «КриптоПро JCP» версия 2.0. Запустите ComLine с вызовом нужного класса либо сам класс, используя следующие параметры командной строки:

`java ComLine NameofClass args` или `java NameofClass args`

например:

```
java ComLine KeyPairGen -alias name_of_key -dname
CN=autor,OU=Security,O=CryptoPro,C=RU -reqCertpath C:/req.txt
```

или

```
java KeyPairGen -alias name_of_key -dname CN=autor,OU=Security,O=CryptoPro,C=RU
-reqCertpath C:/req.txt
```

При этом выполняются: генерирование ключевой пары (в соответствие алгоритмам обмена Диффи-Хелмана и подписи ГОСТ Р 34.10-2001) и соответствующего ей самоподписанного сертификата, запись их на носитель, генерация запроса (DER) на сертификат и запись его в файл, получение сертификата из запроса, представленного в DER-кодировке и запись его в хранилище и в файл, построение цепочки сертификатов.

4.1.3.Работа с ключами обмена и сертификатами аутентификации

Сам процесс обмена начинается с того, что определяется, какая именно пара (закрытый ключ обмена - сертификат аутентификации сервера) будет использован. Для этого пользователю, выполняющему роль сервера, необходимо указать, из какого ключевого хранилища будет прочитана требуемая пара. Указывается это при помощи системных настроек следующим образом:

```
System.setProperty("javax.net.ssl.keyStoreType","HDImageStore");
System.setProperty("javax.net.ssl.keyStorePassword","password");
```

Настройка `"javax.net.ssl.keyStoreType"` задает тип ключевого носителя, с которого будет прочитан закрытый ключ и соответствующий ему сертификат аутентификации (цепочка сертификатов). Таким образом, в качестве типа носителя нужно указывать тот носитель, на который предварительно была записана требуемая пара. Если такую настройку не производить, то по умолчанию в качестве носителя будет использован жесткий диск (`"HDImageStore"`).

Настройка `"javax.net.ssl.keyStorePassword"` определяет пароль на закрытый ключ обмена. Таким образом, в качестве пароля нужно указывать тот пароль, с которым на носитель была записана требуемая ключевая пара. Если такую настройку не производить, то по умолчанию будет использоваться нулевой пароль.

Таким образом, с заданного носителя `"javax.net.ssl.keyStoreType"` будут прочитаны все хранящиеся на нем закрытые ключи и сертификаты (цепочки сертификатов), удовлетворяющие паролю `"javax.net.ssl.keyStorePassword"` и требованиям:

- закрытый ключ соответствует алгоритмам обмена Диффи-Хэллмана и подписи ГОСТ 34.10-2001;
- сертификат имеет следующие расширения: расширение "Использования ключа" включает в себя "Шифрование ключей" или "Согласование ключей"; расширение "Улучшенный ключ" имеет значение "Проверка подлинности сервера".

В качестве рабочей пары будет выбрана первая.

4.1.4.Работа с доверенными сертификатами

В случае двусторонней аутентификации, клиент присылает серверу свой сертификат аутентификации. Сервер при этом должен проверить, что сертификату клиента можно доверять. Для этих целей в ПКЗИ «КриптоПро JTLS» версия 2.0 используется так называемое множество доверенных сертификатов, которое определяется при помощи следующих системных настроек:

```
System.setProperty("javax.net.ssl.trustStoreType","HDImageStore");
```

```
System.setProperty("javax.net.ssl.trustStore","C:\\Java\\jcp\\trust");

System.setProperty("javax.net.ssl.trustStorePassword","password");
```

Настройка "javax.net.ssl.trustStoreType" определяет тип хранилища сертификатов. В «КриптоПро JCP» версия 2.0 хранилище сертификатов, как правило, ассоциируется с некоторым ключевым носителем (при этом тип хранилища сертификатов совпадает с типом ключевого носителя). Таким образом, если в данной настройке в качестве типа хранилища указывает тип ключевого носителя, то с указанного носителя будут прочитаны все корневые сертификаты цепочек и эти сертификаты будут добавлены во множество доверенных сертификатов (если цепочка состоит из одного сертификата, то этот сертификат также будет считаться доверенным). Если такую настройку не производить, то по умолчанию в качестве типа хранилища сертификатов будет использован жесткий диск ("HDImageStore")

Настройка "javax.net.ssl.trustStore" задает путь к хранилищу сертификатов, соответствующему типу "javax.net.ssl.trustStoreType". Такое хранилище используется в том случае, когда сертификат клиента подписан некоторым центром сертификации, корневой сертификат которого не участвует ни в одной цепочке, прочитанной с носителя "javax.net.ssl.trustStoreType", однако серверу известно, что такому центру сертификации можно доверять. Для этого, серверу предварительно необходимо положить корневой сертификат этого центра в хранилище сертификатов "javax.net.ssl.trustStore" следующим образом:

```
String certPath = "C:\\certificate.cer";        // файл, в который был
предварительно
                                                 // сохранен корневой сертификат
центра



KeyStore ks = KeyStore.getInstance("HDImageStore");

ks.load(null, null);

CertificateFactory cf = CertificateFactory.getInstance("X509");


FileInputStream fis = new FileInputStream(certPath);

java.security.cert.Certificate cert = cf.generateCertificate(new
BufferedInputStream(fis));

ks.setCertificateEntry("certificate", cert);


FileOutputStream fos = null;

fos = new FileOutputStream("C:\\Java\\jcp\\trust");

ks.store(fos, "password".toCharArray());

fos.close();
```

Из указанного хранилища сертификатов будут прочитаны все сертификаты, и они также будут добавлены во множество доверенных сертификатов. Если данную настройку не производить, то по умолчанию хранилище сертификатов использоваться не будет (в качестве доверенных будут использоваться только корневые сертификаты цепочек, прочитанных с носителя "javax.net.ssl.trustStoreType").

Настройка "javax.net.ssl.trustStorePassword" определяет пароль на доступ к хранилищу сертификатов (пароль, с которым это хранилище было сохранено). Если такую настройку не производить, то пароль считается нулевым.

Повторим, что определенное таким образом множество доверенных сертификатов сервером используется только в случае двусторонней аутентификации (т.е. в этом случае оно не должно быть пустым). В случае двусторонней аутентификации сервер отравляет

клиенту список имен издателей, которым он доверяет. Этот список формируется из имен субъектов всех доверенных сертификатов.

При получении сертификата аутентификации (цепочки сертификатов) от клиента, он считается успешно проверенным в одном из четырех случаев:

- сертификат аутентификации (конечный сертификат цепочки) содержится во множестве доверенных сертификатов;
- клиентом была отправлена цепочка сертификатов, сертификат аутентификации (конечный сертификат) не является доверенным, но доверенным является один из промежуточных сертификатов или корневой сертификат цепочки. Тогда осуществляется проверка цепочки от конечного до такого доверенного сертификата. Если цепочка проверена, то и сертификат считается проверенным;
- ни один из сертификатов цепочки не является доверенным, но имя издателя корневого, но при этом не самоподписанного, сертификата цепочки содержится в списке доверенных имен. Тогда осуществляется проверка цепочки сертификатов, состоящей из переданной цепочки и доверенного сертификата, имя субъекта которого совпадает с именем издателя переданного корневого. Если цепочка проверена, то и сертификат считается проверенным;
- в противном случае, осуществляется попытка построения цепочки сертификатов на основе переданной цепочки и всего множества доверенных сертификатов. Если цепочка построена, то сертификат считается проверенным.

4.2.Управление ключами и сертификатами клиентом

4.2.1.Общие положения

С точки зрения роли клиента основное различие в управлении ключами проводится для эфемерального и неэфемерального обмена. Неэфемеральный обмен в ПКЗИ «КриптоПро JTLS» версия 2.0 допустим только для ключей обмена, соответствующих алгоритмам обмена Диффи-Хэллмана и подписи ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012. Во всех остальных случаях, эфемеральный обмен производится на ключах, соответствующих алгоритму ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012.

Напомним, что эфемеральный обмен осуществляется в двух случаях:

- в случае односторонней аутентификации, когда закрытый ключ обмена клиента вообще не используется;
- в случае двусторонней аутентификации, когда закрытый ключ обмена клиента не соответствует по параметрам переданному сервером открытому ключу, но при этом создается электронная подпись на закрытом ключе обмена.

Как следует из описанного выше, в случае двусторонней аутентификации, если данная сторона является клиентом, необходимо чтобы у нее существовали закрытый ключ и соответствующий ему сертификат (цепочка сертификатов) аутентификации. Причем, допустимы к использованию только следующие пары (закрытый ключ обмена и соответствующий ему сертификат аутентификации):

- закрытый ключ соответствует алгоритмам обмена Диффи-Хэллмана и подписи ГОСТ 34.10-2001 или ГОСТ Р 34.10-2012;
- сертификат имеет следующие расширения: расширение "Использования ключа" включает в себя "Шифрование ключей" или "Согласование ключей"; расширение "Улучшенный ключ" имеет значение "Проверка подлинности клиента".

4.2.2.Действия перед началом обмена с сервером

Таким образом, перед началом осуществления соединения с сервером в случае двусторонней аутентификации, такие закрытый ключ и соответствующий ему сертификат (цепочку сертификатов) необходимо создать и затем положить в ключевое хранилище, из которого они и будут прочитаны в процессе обмена. Осуществление таких действий производится аналогично описанию для сервера. Единственная разница состоит в том, что создаваемый сертификат аутентификации имеет расширения сертификата аутентификации клиента. Создание таких сертификатов производится по следующей схеме:

```
String keyAlg = "GOST3410DHEL";
```

```
String certName = "CN=newCert, O=CryptoPro, C=RU";

String httpAddress = "http://www.cryptopro.ru/certsrv/";


// создание запроса на сертификат аутентификации сервера

GostCertificateRequest request = new GostCertificateRequest();

request.setKeyUsage(GostCertificateRequest.CRYPT_DEFAULT);

request.addExtKeyUsage(GostCertificateRequest.INTS_PKIX_CLIENT_AUTH);

request.setPublicKeyInfo(pubKey);

request.encodeAndSign(privKey);


// отправка запроса центру сертификации и получение от центра

// сертификата в DER-кодировке

byte[] encoded = request.getEncodedCert(httpAddress);


// генерация X509-сертификата из закодированного представления сертификата

CertificateFactory cf = CertificateFactory.getInstance("X509");

java.security.cert.Certificate cert = cf.generateCertificate(new
ByteArrayInputStream(encoded));
```

4.2.3.Работа с ключами обмена и сертификатами аутентификации

Эта работа осуществляет только в случае двусторонней аутентификации и аналогична описанию для сервера. Единственная разница состоит в выборе пары (закрытый ключ обмена - сертификат аутентификации). Если в случае сервера, в качестве рабочей пары выбирается первая подходящая пара, прочитанная с ключевого носителя, то в данном случае возможны два варианта:

- если на носителе существует подходящий сертификат (цепочка сертификатов) и при этом удовлетворяющий переданному сервером списку доверенных имен издателей, а открытый ключ этого сертификата совпадает по параметрам с открытым ключом сертификата сервера, то этот сертификат будет передан серверу, а на соответствующем ему закрытом ключе будет осуществлен обмен;
- в противном случае, будет выбран первый подходящий сертификат, создана эфемеральная ключевая пара с параметрами открытого ключа сервера и отправлено сообщение CERTIFICATE_VERIFY в соответствии с созданной эфемеральной парой и выбранным сертификатом.

4.2.4.Работа с доверенными сертификатами

Поскольку считается, что сервер ВСЕГДА присылает свой сертификат клиенту, то проверка сертификата клиентом осуществляется всегда. Она основывается на множестве доверенных сертификатов клиента, формируемого и используемого аналогично описанию сервера. Очевидно, что это множество должно быть не пустым.

5. Реализуемые шифр-сюиты и совместимость по ним с различными версиями КриптоПро CSP

5.1. Описание реализуемых «КриптоПро JTLS» версия 2.0 шифр-сюит

ПКЗИ «КриптоПро JTLS» версия 2.0 реализует три варианта шифр-сюит. В реализованных шифр-сюитах используются следующие алгоритмы:

- алгоритм ключевого обмена Диффи-Хэллмана в соответствии с алгоритмом электронной подписи ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012.
- алгоритм хэширования в соответствии с ГОСТ Р 34.11-94 или ГОСТ Р 34.11-2012;
- алгоритм выработки электронной подписи в соответствии с ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012. Выработка ЭП осуществляется в том случае, когда данная сторона является клиентом, требуется ее аутентификация, но параметры ключа ЭП клиента не соответствуют параметрам ключа проверки ЭП сервера;
- алгоритм проверки электронной подписи в соответствии с ГОСТ Р 34.10-2001 или ГОСТ Р 34.10-2012. Проверка ЭП осуществляется в том случае, когда данная сторона является сервером, требовалась аутентификация клиента, но параметры ключа ЭП клиента не соответствовали параметрам ключа проверки ЭП сервера;
- алгоритм шифрования соединения в соответствии с ГОСТ 28147-89;
- алгоритм имитозащиты передаваемых данных в соответствии с ГОСТ 28147-89.

Ниже приводится таблица с кратким описанием реализуемых «КриптоПро JTLS» версия 2.0 шифр-сюит:

Имя шифр-сюиты	Идентификатор шифр-сюиты	Алгоритм ключей обмена	Режим шифрования данных по алгоритму ГОСТ Р 28147-89
TLS_CIPHER_2012	0xff85	ГОСТ 34.10-2012	Гаммирование
TLS_CIPHER_2001	0x81	ГОСТ 34.10-2001	Гаммирование
SSL3_CK_GVO_KB2	0x32	ГОСТ 34.10-2001	Гаммирование с обратной связью
SSL3_CK_GVO	0x31	ГОСТ 34.10-2001	Гаммирование с обратной связью

В данной таблице шифр-сюиты перечислены в порядке уменьшения приоритета. Соответственно, при получении от клиента списка шифр-сюит, сервер выбирает подходящую шифр-сюиту с наибольшим приоритетом.

5.2. Поддержка шифр-сюит клиентом и сервером в «КриптоПро JTLS» версия 2.0

При инициализации процесса обмена, текущая сторона формирует список поддерживаемых шифр-сюит. Этот список будет различаться в зависимости от того, является текущая сторона клиентом или сервером. Поддерживаемые шифр-сюиты заносятся в список в соответствии с порядком их приоритета (первой заносится шифр-сюита с наибольшим приоритетом). Дальнейший процесс обмена текущей стороной будет осуществляться в соответствии с сформированным списком. Отправка клиентом списка серверу осуществляется именно в таком виде, в каком он был сформирован (в порядке уменьшения приоритета). Разбор полученного от клиента списка сервером также осуществляется в порядке уменьшения приоритета.

- **TLS_CIPHER_2012** - поддерживается обеими сторонами. Если данная сторона является сервером, и в полученном ею списке поддерживаемых клиентом шифр-сюит содержится *TLS_CIPHER_2012*, то именно она и выбирается в качестве рабочей. Если данная сторона является клиентом, то шифр-сюита *TLS_CIPHER_2012* отправляется первой в списке поддерживаемых.
- **TLS_CIPHER_2001** - поддерживается обеими сторонами. Если данная сторона является сервером, и в полученном ею списке поддерживаемых клиентом шифр-сюит содержится *TLS_CIPHER_2001*, то именно она и выбирается в качестве рабочей. Если данная сторона является клиентом, то шифр-сюита *TLS_CIPHER_2001* отправляется первой в списке поддерживаемых.

- **SSL3_CK_GVO_KB2** - поддерживается обеими сторонами. Если данная сторона является сервером, и в полученном ею списке поддерживаемых клиентом шифр-сюит не содержалось *TLS_CIPHER_2001*, то в качестве рабочей выбирается *SSL3_CK_GVO_KB2* (при условии, что она содержится в списке). Если данная сторона является клиентом, то шифр-сюита *SSL3_CK_GVO_KB2* отправляется второй в списке поддерживаемых.
- **SSL3_CK_GVO** - поддерживается сервером и опционально клиентом (по умолчанию клиентом не поддерживается). Если данная сторона является сервером, и в полученном ею списке поддерживаемых клиентом шифр-сюит не содержалось ни *TLS_CIPHER_2001*, ни *SSL3_CK_GVO_KB2* , то в качестве рабочей выбирается *SSL3_CK_GVO*. Если данная сторона является клиентом, то по умолчанию она данную шифр-сюиту не поддерживает. Следовательно, данная шифр-сюита не заносится в отправляемый серверу список, а если в ответ на переданный список сервер ответил данной шифр-сюитой, то считается, что обмен не может быть осуществлен. *Однако, для поддержки старых серверов (например, DIGT), реализована возможность опционального подключения шифр-сюиты SSL3_CK_GVO в список, поддерживаемых клиентом шифр-сюит. Данное подключение может быть осуществлено при помощи системной настройки* `System.setProperty("javax.net.ssl.supportGVO","true")`. *Такое подключение не рекомендуется к использованию. После того, как шифр-сюита подключена, она будет занесена в список поддерживаемых клиентом шифр-сюит последней.*

Таким образом, формируются следующие списки поддерживаемых шифр-сюит:

Клиент	**Сервер**
TLS_CIPHER_2012	TLS_CIPHER_2012
TLS_CIPHER_2001	TLS_CIPHER_2001
SSL3_CK_GVO_KB2	SSL3_CK_GVO_KB2
SSL3_CK_GVO (опционально)	SSL3_CK_GVO

5.3.Совместимость по шифр-сюитам с различными версиями КриптоПро CSP

Ниже приводится таблица, в которой описывается, какая именно шифр-сюита будет выбрана сервером при условии при осуществлении процесса обмена с различными версиями «КриптоПро JTLS» версия 2.0.

Клиент	**Сервер**	**Выбираемая сервером шифр-сюита**
JTLS 2.0	JTLS 2.0	TLS_CIPHER_2012, TLS_CIPHER_2001
JTLS 2.0/ CSP 4.0	CSP 4.0/ JTLS 2.0	TLS_CIPHER_2012, TLS_CIPHER_2001
JTLS 2.0/ CSP 3.6	CSP 3.6/ JTLS 2.0	TLS_CIPHER_2001
JTLS 2.0	CSP 2.0	SSL3_CK_GVO_KB2
JTLS 2.0	CSP 3.0	TLS_CIPHER_2001
CSP 2.0	JTLS 2.0	SSL3_CK_GVO
CSP 3.0	JTLS 2.0	TLS_CIPHER_2001

6. Контрольная панель

Основной набор закладок контрольной панели «КриптоПро JCP» версия 2.0 описан в Руководстве администратора. После установки модуля «КриптоПро JTLS» версия 2.0 на контрольной панели появятся соответствующие закладки.

6.1.Закладка "Сервер JTLS"

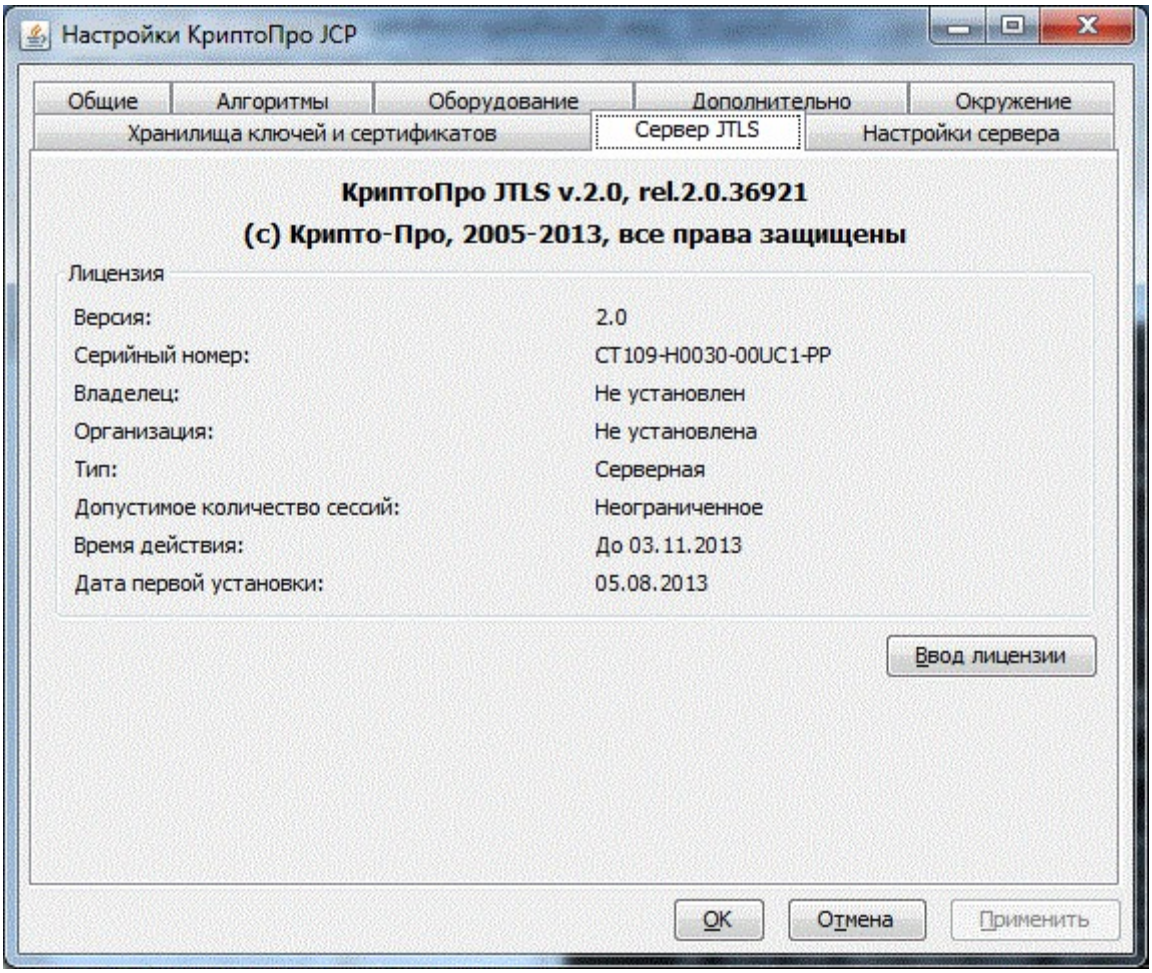

Рисунок 1. Внешний вид панели "Лицензия" (временная лицензия)

Данная панель содержит информацию о серверной лицензии «КриптоПро JTLS» версия 2.0. Работа с данной панелью аналогична работе с закладкой "Общие" (панель "Лицензия").

6.2.Закладка "Настройки сервера"

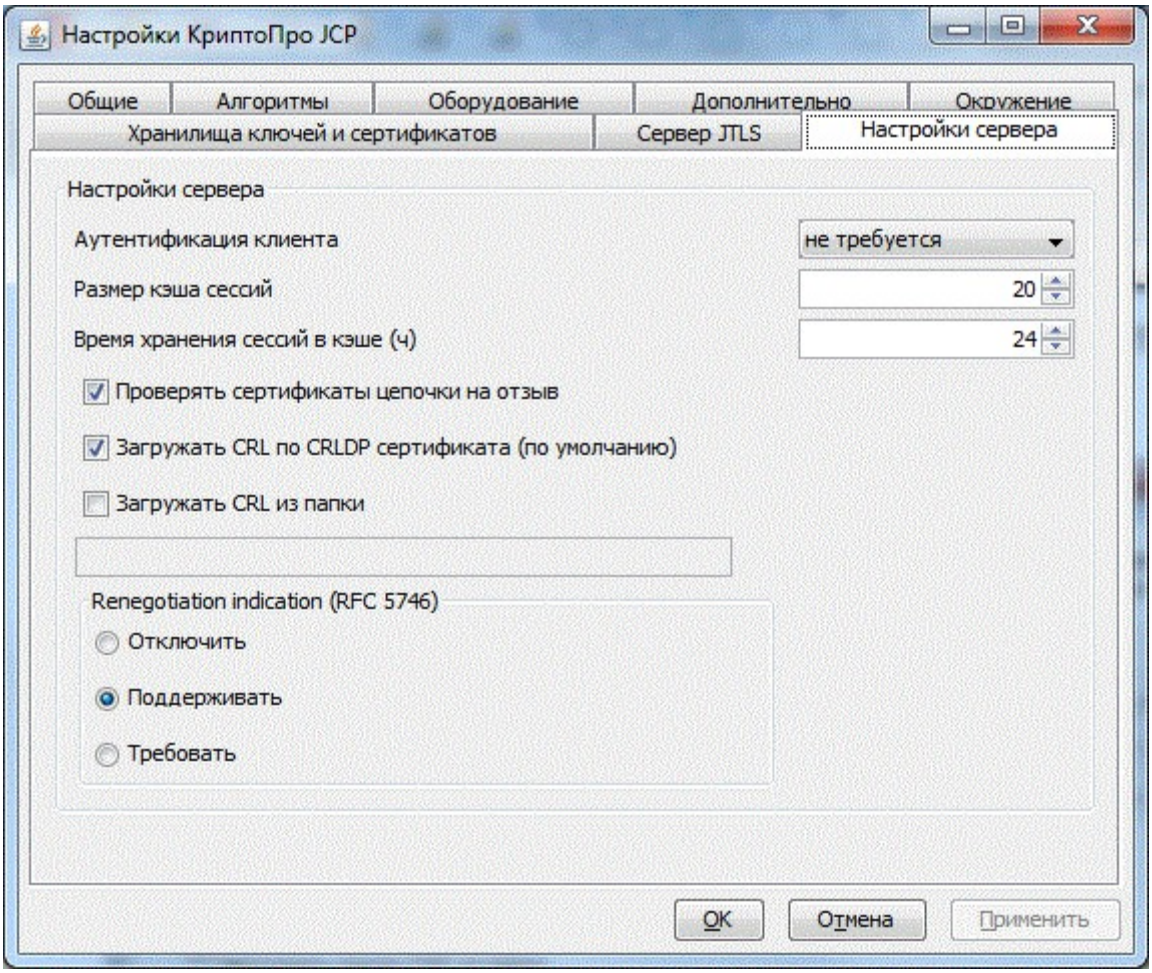

Рисунок 2. Внешний вид панели "Настройки сервера"

Данная панель содержит настройки сервера:

- Аутентификация клиента:
  - не требуется (по умолчанию)
  - желательна
  - требуется
- Размер кэша сессий (количество сессий; по умолчанию 0 - неограниченное)
- Время хранения сессий в кэше (по умолчанию 24 часа; если размер кэша сессий не задан (=0), то "старые" сессии удаляться не будут).
- Возможность полного отключения проверки цепочки сертификатов на отзыв, включение проверки с условием загрузки СОС из сети по CRLDP сертификата, включение проверки с условием загрузки СОС из папки (задается абсолютный путь к папке с СОС).
- Отключение, включение и требование поддержки расширения Renegotiation Indication (RFC 5746). Задание данных настроек с помощью параметров *-Dru.CryptoPro.ssl.allowUnsafeRenegotiation=<value>* *-Dru.CryptoPro.ssl.allowLegacyHelloMessages=<value>* в приложении имеет приоритет выше и переопределяет настройки «КриптоПро JTLS» версия 2.0. Пары указанных свойств образуют следующие группы:

Режим	Allow Legacy Hello Messages	Allow Unsafe Renegotiation	Аналогия с CSP TLS
Строгий (strict)	false	false	Требуем RFC 5746: Наличие RI обязательно, проверка выполняется
Безопасный (interoperable)	true (SUN default)	false	Поддерживаем RFC

			5746 (по умолчанию в CryptoPro CSP 4.0): наличие RI необязательно, проверка может выполняться
Небезопасный (insecure)	true	true	Не поддерживаем RFC 5746 (по умолчанию в CryptoPro CSP): наличие RI необязательно, проверка не выполняется

7. Использование «КриптоПро JTLS» версия 2.0 в Apache Tomcat

Основы использования SSL/TLS в читайте в Apache Tomcat SSL Configuration HOW-TO.

Apache Tomcat имеет собственный интерфейс для встраивания SSL/TLS протоколов.

Настройка Apache Tomcat проводится в следующем порядке:

1. Установка «КриптоПро JTLS» версия 2.0, как указано выше.
2. Создание сертификата "Проверки подлинности сервера".
3. Настройка коннектора Apache Tomcat.

Важным обстоятельством является тот факт, что если включена проверка цепочки сертификатов на отзыв по CRLDP сертификата, то необходимо разрешить загрузку COC, задав параметры

```
System.setProperty("com.sun.security.enableCRLDP", "true");
```

либо

```
System.setProperty("com.ibm.security.enableCRLDP", "true");
```

7.1. Создание сертификата Apache Tomcat

Сертификат выпускается, как указано выше или в руководстве программиста. Его общее имя (common name) должно совпадать с DNS-именем (или IP - при обращении клиентов **только** по IP) сервера Apache Tomcat. Apache Tomcat перебирает все сертификаты пользователя, под учетной записью которого он работает, пока не найдет подходящий (по назначению и паролю), поэтому необходимо обеспечить его уникальность именно в этом смысле.

По умолчанию (в Windows) Apache Tomcat ищет контейнер с сертификатом под учетной записью системы (даже если в настройках Apache Tomcat указать свойство LogOn под пользовательской учетной записью, поиск подходящего контейнера он все равно ведет в этой папке) поэтому следует либо создать его под учетной записью системы, либо вручную переложить созданный под своей учетной записью контейнер из

```
%USERPROFILE%\Local Settings\Application Data\Crypto Pro\
```

в

```
%USERPROFILE%\..\Default User\Local Settings\Application Data\Crypto Pro\,
```

либо настроить Apache Tomcat под свою учетную запись следующим образом:

Панель управления -> Администрирование -> Службы и приложения -> Службы -> Apache Tomcat свойства -> вход в систему: с учетной записью (указать имя и пароль).

7.2. Настройка коннектора Apache Tomcat

Для настройки коннектора Apache Tomcat необходимо добавить в файл `<CATALINA_HOME>/conf/server.xml` строки по следующему образцу:

```
<Connector port="8443" maxHttpHeaderSize="8192"
           protocol="org.apache.coyote.http11.Http11NioProtocol"
           SSLEnabled="true"
           maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
           enableLookups="false" disableUploadTimeout="true"
           acceptCount="100" scheme="https" secure="true"
           clientAuth="false"
           sslProtocol="GostTLS"
```

```
                algorithm="GostX509"
                keystoreProvider="JCP"
                keystoreFile="<USER_HOME>/.keystore"
                keystorePass="11111111"
                keystoreType="HDImageStore"
                keyalg="GOST3410EL"
                sigalg="GOST3411withGOST3410EL"

        />
```

Описание основных параметров смотрите в документации на Apache Tomcat. В `keystoreFile` указывается путь к соответствующему файлу. В `keystorePass` - пароль на контейнер. При указанном выше описании используется встроенный переходник для JSSE.

Важно отметить, что для Apache Tomcat версии 7.0.42 и выше может потребоваться указание дополнительных параметров:

```
ciphers="TLS_CIPHER_2001"
sslEnabledProtocols="GostTLS"
```

Tomcat может использовать библиотеки Apache Portable Runtime (APR) для повышения производительности. APR использует платформо-зависимую реализацию SSL, которая не может работать с российскими ГОСТ алгоритмами и Java настройками типа `keystoreFile` и выдаст ошибку. Для предотвращения автоконфигурации через APR и задания Java коннектора, независимо от того загружены ARP библиотеки или нет, необходимо в описании коннектора явно задать имя класса реализации в атрибуте протокола `protocol="org.apache.coyote.http11.Http11NioProtocol"`.

При аутентификации клиента (`clientAuth="true"`) следует также указать путь к хранилищу сертификатов (`truststoreFile="<USER_HOME>/.keystore"`) и пароль (`truststorePass="1111111".`)

7.3.Настройка журналирования «КриптоПро JTLS» версия 2.0 в Apache Tomcat

Для настройки журналирования действий «КриптоПро JTLS» версия 2.0 в файле `<CATALINA_HOME>/conf/logging.properties` необходимо:

- добавить новый handler в строку `handlers = ...`
- добавить описание его свойств в соответствующем разделе
- добавить описание ru.CryptoPro.ssl.SSLLogger

Соответствующие изменения помечены жирным шрифтом:

```
      handlers = 1catalina.org.apache.juli.FileHandler,
2localhost.org.apache.juli.FileHandler, 3manager.org.apache.juli.FileHandler,
4admin.org.apache.juli.FileHandler, 5host-manager.org.apache.juli.FileHandler,
6jtls.org.apache.juli.FileHandler, java.util.logging.ConsoleHandler


      .handlers = 1catalina.org.apache.juli.FileHandler,
java.util.logging.ConsoleHandler


      ############################################################
      # Handler specific properties.
      # Describes specific configuration info for Handlers.
      ############################################################


      1catalina.org.apache.juli.FileHandler.level = FINE
```

```
1catalina.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.FileHandler.prefix = catalina.

2localhost.org.apache.juli.FileHandler.level = FINE
2localhost.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
2localhost.org.apache.juli.FileHandler.prefix = localhost.

3manager.org.apache.juli.FileHandler.level = FINE
3manager.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
3manager.org.apache.juli.FileHandler.prefix = manager.

4admin.org.apache.juli.FileHandler.level = FINE
4admin.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
4admin.org.apache.juli.FileHandler.prefix = admin.

5host-manager.org.apache.juli.FileHandler.level = FINE
5host-manager.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
5host-manager.org.apache.juli.FileHandler.prefix = host-manager.

6jtls.org.apache.juli.FileHandler.level = FINE
6jtls.org.apache.juli.FileHandler.directory = ${catalina.base}/logs
6jtls.org.apache.juli.FileHandler.prefix = jtls.

java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter



############################################################
# Facility specific properties.
# Provides extra control for each logger.
############################################################

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers           =
2localhost.org.apache.juli.FileHandler


org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].level    =
INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/manager].handlers
= 3manager.org.apache.juli.FileHandler


org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/admin].level      =
INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/admin].handlers   =
4admin.org.apache.juli.FileHandler


org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/host-
manager].level = INFO
```

```
        org.apache.catalina.core.ContainerBase.[Catalina].[localhost].[/host-
manager].handlers = 5host-manager.org.apache.juli.FileHandler


        ru.CryptoPro.ssl.SSLLogger.level = FINE
        ru.CryptoPro.ssl.SSLLogger.handlers = 6jtls.org.apache.juli.FileHandler


        # For example, set the com.xyz.foo logger to only log SEVERE
        # messages:
        #org.apache.catalina.startup.ContextConfig.level = FINE
        #org.apache.catalina.startup.HostConfig.level = FINE
        #org.apache.catalina.session.ManagerBase.level = FINE
```

8. Отладка

Для отладки ssl соединения можно воспользоваться встроенным логгером. Для этого установите уровень FINE в jre/lib/logging.properties:

```
.level = FINE
...
java.util.logging.ConsoleHandler.level = FINE
```

SSLLogger является расширением стандартного класса Logger (см. документацию java).