127018, Москва, Сущёвский Вал, 18
Телефон: (495) 995 4820
Факс: (495) 995 4820
https://CryptoPro.ru
E-mail: info@CryptoPro.ru

КРИПТОПРО

Средство Криптографической Защиты Информации

КриптоПро CSP Версия 5.0 KC2 2-Base

Руководство программиста JavaTLS

ЖТЯИ.00102-01 96 03
Листов 23

2019 г

Содержание

Аннотация

Настоящий документ описывает состав функций и тестовое ПО программного комплекса защиты информации (далее — ПКЗИ) КриптоПро JavaTLS и предназначен для разработки прикладного ПО с непосредственным вызовом функций ПКЗИ, а также определяет требования к операционным системам при встраивании СКЗИ КриптоПро CSP.

1 Введение

КриптоПро JavaTLS реализует протоколы SSL и TLS в соответствии с российскими криптографическими алгоритмами, предоставляемыми криптопровайдером КриптоПро JavaCSP (СКЗИ КриптоПро CSP 5.0 КС2).

Основные функции, реализуемые КриптоПро JavaTLS:

• две схемы аутентификации с использованием обмена ключей по алгоритмам Диффи-Хэллмана и хэширования в соответствии с ГОСТ Р 34.11-94 и 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 JavaTLS

В случае использования Java-машин версии 10 и выше:

Установка модуля КриптоПро JavaTLS не требуется, функционал провайдера будет доступен после добавления модуля в classpath.

Эксплуатация осуществляется путем добавления провайдера в список java.security:

```
security.provider.<N>=JTLS
```

или программно, с помощью Security.addProvider:

```
Security.addProvider(new Provider()); // провайдер JTLS
```

Перед тем, как приступить к установке КриптоПро JavaTLS, необходимо установить криптопровайдер КриптоПро JavaCSP, включая модули шифрования.

Для запуска программы установки необходимо вызвать Java с именем jar-файла, например:

```
<JRE>/bin/java -jar cpSSL.jar
```

Также возможна установка с вводом серийного номера с помощью класса ru.CryptoPro.ssl.JTLSInstall:

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

Установка модуля так же может быть выполнена с помощью графического (setup.exe, setup_gui.sh) или

консольного (setup_console.bat, setup_console.sh) инсталляторов.

Подробную информацию по установке КриптоПро JavaTLS см. в «ЖТЯИ.00102-01 92 05. КриптоПро CSP. Инструкция по использованию JavaTLS».

Использование КриптоПро JavaTLS возможно только на Java 1.7 и 1.8. Для преодоления экспортных ограничений на стойкую криптографию см. Особенности подключения КриптоПро JavaTLS.

3 Работа через внешний интерфейс JSSE

ПКЗИ КриптоПро JavaTLS реализует стандартный интерфейс Java Secure Socket Extension (JSSE) v.1.7 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 JavaTLS

По причине включения в JSSE помимо самого интерфейса некоторых его реализаций (примером такой реализации на виртуальной машине SUN является провайдер `com.sun.net.ssl.internal.ssl.Provider`), существуют некоторые особенности использования ПКЗИ КриптоПро JavaTLS.

Возможна ситуация, когда установленная JRE имеет экспортные ограничения. США запрещает экспорт «сильной» криптографии и JavaCSP 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 можно просто скопировать `US_export_policy.jar` в `local_policy.jar` (оба файла должны присутствовать).

3.2 Использование КриптоПро JavaTLS через внешний интерфейс JSSE

После того, как КриптоПро JavaTLS 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 рассматриваемая 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 рассматриваемая 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
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 дистрибутив КриптоПро JavaCSP. Пакет `ComLine` модуля `Samples` содержит примеры сервера и клиента, запускаемые из командной строки.

Поддерживаемые протоколы:

- `GostTLS` — соответствует TLS v. 1.0
- `GostTLSv1.1` — соответствует TLS v. 1.1
- `GostTLSv1.2` — соответствует TLS v. 1.2

4 Управление ключами и сертификатами

В ПКЗИ КриптоПро JavaTLS 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 JavaTLS 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 или ГОСТ 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 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 сертификатов) необходимо создать и затем положить в ключевое хранилище, из которого они и будут прочитаны в процессе обмена. Методы выполнения этих операций подробно описаны в ЖТЯИ.00102-01 92 06. Руководство программиста JavaCSP. 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("GOST3410DH");
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:

```
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 использовался тестового центра сертификации КриптоПро.

Также можно воспользоваться готовыми классами пакета ComLine из модуля Samples, входящего в состав КриптоПро JavaCSP. 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 происходят 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.

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 или ГОСТ 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 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 JavaTLS 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. КриптоПро JavaCSP 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.

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 ЦС
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.

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 JavaTLS 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.

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 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 = "GOST3410DH";
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 JavaTLS шифр-сюит

ПКЗИ КриптоПро JavaTLS 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.

Краткое описание реализуемых КриптоПро JavaTLS шифр-сюит представлено в табл. 1. 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.

Таблица 1. Описание реализуемых шифр-сюитов

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

5.2 Поддержка шифр-сюит клиентом и сервером в КриптоПро JavaTLS

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 не содержалось TLS_CIPHER_2012, то в качестве рабочей выбирается TLS_CIPHER_2001. Если данная сторона является клиентом, то шифр-сюита TLS_CIPHER_2001 отправляется второй в списке поддерживаемых.

Таким образом, формируются следующие списки поддерживаемых шифр-сюит:

Клиент	Сервер
TLS_CIPHER_2012	TLS_CIPHER_2012
TLS_CIPHER_2001	TLS_CIPHER_2001

5.3 Совместимость по шифр-сюитам с различными версиями КриптоПро CSP

В табл. 2 описывается, какая именно шифр-сюита будет выбрана сервером при осуществлении процесса обмена с различными версиями КриптоПро JavaTLS.

Таблица 2. Совместимость по шифр-сюитам с различными версиями КриптоПро CSP

Клиент	Сервер	Выбираемая сервером шифр-сюита
JTLS 2.0 / CSP 5.0	CSP 5.0 / JTLS 2.0	TLS_CIPHER_2012
JTLS 2.0 / JTLS 2.0	JTLS 2.0 / JTLS 2.0	TLS_CIPHER_2012
JTLS 2.0	JTLS 2.0	TLS_CIPHER_2012
JTLS 2.0 / CSP 5.0	CSP 5.0 / JTLS 2.0	TLS_CIPHER_2012
JTLS 2.0 / CSP 4.0	CSP 4.0 / JTLS 2.0	TLS_CIPHER_2012, TLS_CIPHER_2001

6 Настройка сервера через контрольную панель

После установки КриптоПро JavaTLS на контрольной панели появляются 2 закладки — «Сервер JTLS» и «Настройки TLS». Подробную информацию о них см. в «ЖТЯИ.00102-01 92 05. КриптоПро CSP. Инструкция по использованию JavaTLS».

С помощью закладки «Настройки TLS» устанавливаются следующие настройки сервера:

- необходимость аутентификации клиента (по умолчанию не 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 COC из сети по CRLDP сертификата, включение проверки с условием загрузки COC из папки (задается абсолютный путь к папке с COC);
- отключение, включение и требование поддержки расширения Renegotiation Indication (RFC 5746). Задание данных настроек с помощью параметров `Dru.CryptoPro.ssl.allowUnsafeRenegotiation=<value>` и `Dru.CryptoPro.ssl.allowLegacyHelloMessages=<value>` в приложении имеет более высокий приоритет и переопределяет настройки JavaTLS. Пары указанных свойств образуют следующие группы (см. табл. 3);
- возможность отправки клиентом расширения Renegotiation Indication (по умолчанию разрешено).

Таблица 3. Режимы поддержки Renegotiation Indication (RFC 5746)

Режим	**Allow Legacy Hello Messages**	**Allow Unsafe Renegotiation**	**Аналогия с КриптоПро 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 КриптоПро CSP 5.0 KC2): наличие RI необязательно, проверка может выполняться
Небезопасный (insecure)	true	true	Не поддерживаем RFC 5746 (по умолчанию в КриптоПро CSP 5.0 KC2): наличие RI необязательно, проверка не выполняется

7 Использование КриптоПро JavaTLS в Apache Tomcat

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

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

Настройка Apache Tomcat проводится в следующем порядке:
1) Установка КриптоПро JavaTLS;
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

Для выпуска сертификата следуйте инструкциям, описанным в Управление ключами и сертификатами. Общее имя сертификата (Common Name, CN) 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%\LocalSettings\ApplicationData\CryptoPro\` в `%USERPROFILE%\..\DefaultUser\LocalSettings\ApplicationData\CryptoPro\`, 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 см. 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.

Для того, чтобы настроить tomcat для использования JavaCSP, следует указать следующие параметры:

```
<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="JCSP"
    keystoreFile="<USER_HOME>/.keystore"
    keystorePass="11111111"
    keystoreType="<TYPE>"

    trustStoreProvider="JCP"
    trustStoreType="CertStore"
    trustStoreFile="<USER_HOME>/.keystore"
    trustStorePass="11111111"

    keyalg="GOST3410EL"
    sigalg="GOST3411withGOST3410EL"

/>
```

Для работы с контейнерами используется провайдер JavaCSP, в то время как с хранилищем доверенных сертификатов работает JCP (с типом хранилища `CertStore`). `<TYPE>` — обозначает тип контейнера для провайдера JavaCSP, например, при настройке tomcat в ОС Windows TYPE соответствует REGISTRY, а в ОС Linux — HDIMAGE.

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

дополнительных параметров:

```
ciphers="TLS_CIPHER_2001"
sslEnabledProtocols="TLSv1"
```

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").

Важно отметить, что для Apache Tomcat 8.5 и выше может потребоваться специальный модуль-адаптер, чьи библиотека, описание и исходники находятся в папке дистрибутива Doc\WebServerIntegration\Tomcat9.

7.3 Настройка журналирования КриптоПро JavaTLS в Apache Tomcat

Для настройки журналирования действий КриптоПро JavaTLS 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.

Пример настройки файла logging.properties с уровнем **FINE**:

```
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 SSL-соединения

Для отладки SSL-соединения можно воспользоваться встроенным логгером SSLLogger.

Для этого установите уровень FINE в jre/lib/logging.properties:

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

SSLLogger является расширением стандартного класса Java Logger.

9 Особенности криптопровайдера КриптоПро JavaCSP

JCSP (JavaCSP) — провайдер, осуществляющий все криптографические операции путем обращения к КриптоПро CSP. Криптопровайдер КриптоПро JavaCSP реализует стандартный интерфейс Java Cryptography Architecture (JCA) в соответствии с российскими криптографическими алгоритмами.

Для указания имени провайдера вместо `JCP.PROVIDER_NAME` следует использовать `JCSP.PROVIDER_NAME`. Для того, чтобы использовать установленный провайдер JavaCSP в cpSSL, перед выполнением кода следует указать следующую команду:

```
cpSSLConfig.setDefaultSSLProvider(JCSP.PROVIDER_NAME); // класс cpSSLConfig входит в
                                                       // пакет ru.CryptoPro.ssl.util
```

либо

```
System.setProperty("ru.CryptoPro.defaultSSLProv", "JCSP");
```

9.1 Загрузка одного экземпляра ключевого контейнера с помощью класса StoreInputStream

По умолчанию менеджер ключей модуля cpSSL осуществляет подбор ключевых контейнеров простым перечислением их с помощью метода `aliases()` класса `KeyStore` и применением к ним некоторого пароля, который может быть передан как при создании объекта `SSLContext`, так и путем задания свойства `keyStorePassword`.

Однако этот способ подбора контейнера имеет некоторые негативные последствия в JavaCSP, поэтому более правильным является загрузка одного заранее известного ключевого контейнера. Это возможно с помощью класса `ru.CryptoPro.JCP.KeyStore.StoreInputStream`. Класс описан в javadoc-документации дистрибутива. Конструктор класса принимает название контейнера (алиас) в двух форматах: алиас ключа и FQCN-имя контейнера.

Например, можно создать SSL-контекст следующим образом:

```
// Формируем поток с именем контейнера
InputStream container = new StoreInputStream("alias");
// Формируем объект для загрузки контейнера
KeyStore keyStore = KeyStore.getInstance(JCSP.HD_STORE_NAME);
keyStore.load(container, null); // Загружаем контейнер
KeyManagerFactory kmf = KeyManagerFactory.getInstance("GostX509");
kmf.init(keyStore, "password".toCharArray()); // Инициализация менеджера ключей
SSLContext sslCtx = SSLContext.getInstance("GostTLS"); // Подготовка SSL-контекста
sslCtx.init(kmf.getKeyManagers(), ... , null); // Создание SSL-контекста
```

либо воспользоваться свойствами keyStore:

```
System.setProperty("javax.net.ssl.keyStoreType", "HDIMAGE");
System.setProperty("javax.net.ssl.keyStore", "alias");
```

и передать в него алиас контейнера.

При загрузке контейнера с помощью класса StoreInputStream все функции из KeyStore будут возвращать ключ, сертификат и другие данные только из указанного контейнера, за исключением функций типа SetXXX, которые предполагают сохранение контейнера.

Особенностью использования StoreInputStream является также то, что можно получить закрытый ключ, сертификат и другую информацию, вызвав getXXX с null вместо алиаса, например:

```
// Получение закрытого ключа
InputStream PrivateKey pk = keyStore.getKey(null, "password");
// Получение списка из одного контейнера
InputStream Enumeration aliases = keyStore.aliases();
// Получение сертификата
InputStream Certificate cert = keyStore.getCertificate(null);
```

Алиас в этом случае не требуется, т.к. предполагается, что он уже был передан ранее с помощью класса StoreInputStream. Однако, данное правило не распространяется на функцию getEntry, т.к. ей запрещено передавать null вместо алиаса.