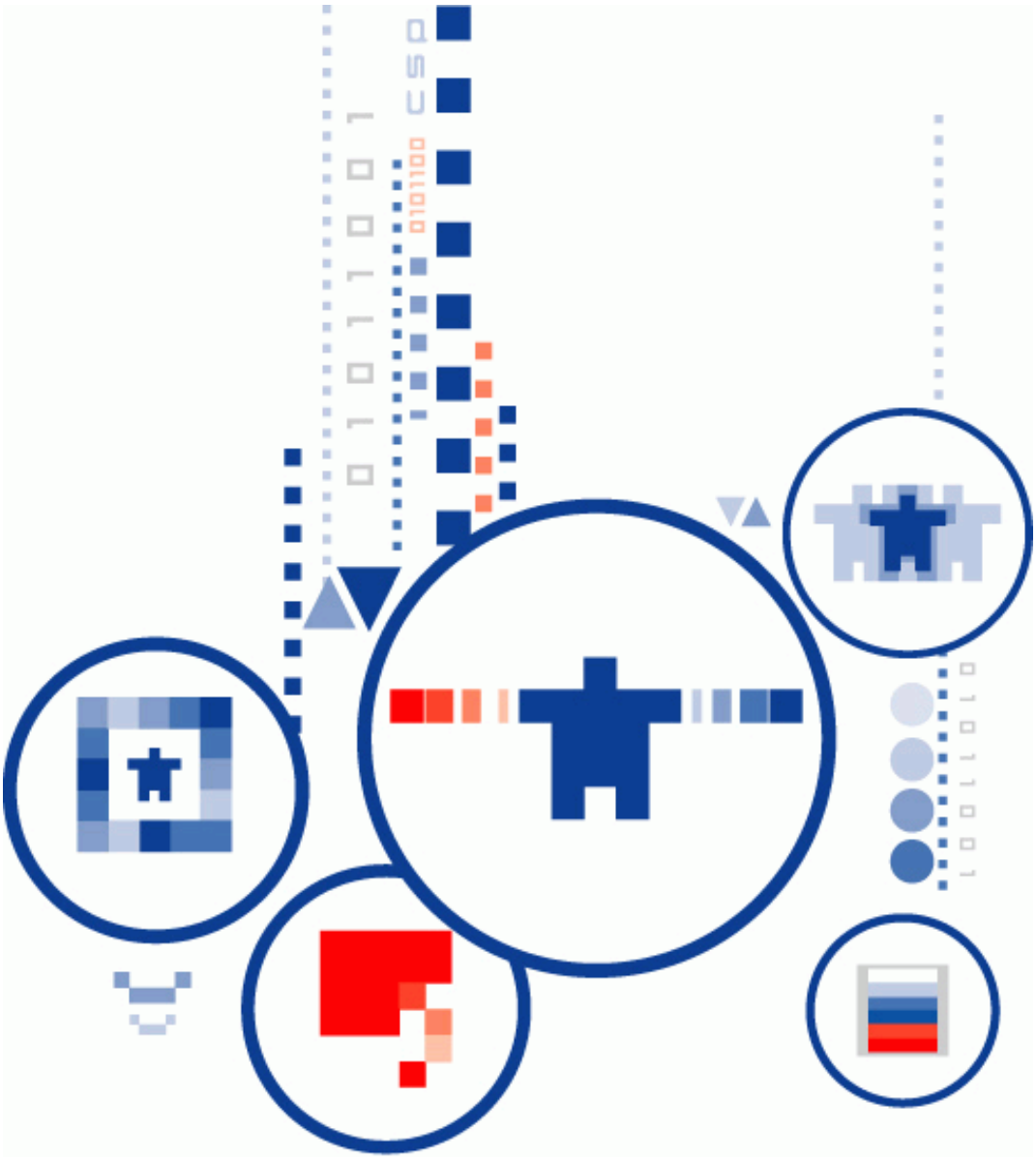

Сервер Электронной Подписи

«КриптоПро DSS»

Компонент ПАКМ «КриптоПро HSM»

myDSS SDK. Руководство разработчика.

Android

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

DSSUser	—	Объект, содержащий всю необходимую информацию для выполнения действий от имени пользователя. Содержит в том числе информацию для доступа к экземпляру DSS (url, корневой сертификат и пр.), "вектор аутентификации" для подтверждения действий и пр. С точки зрения DSS данный объект является устройством, подключенным к учетной записи пользователя DSS.
DSSDevice	—	Объект, содержащий информацию об устройствах, подключенных к той же учетной записи, что и объект DSSUser. Данный объект не содержит "вектор аутентификации" и не может использоваться для подтверждения операций или выполнения других действий.
DSSOperation	—	Операция DSS, для которой требуется подтверждение/отклонение. Операция может содержать несколько документов. При подтверждении/отклонении операции все содержащиеся в ней документы будут подписаны/отклонены. Операция создается на сервере DSS.
DSSOperation.DSSDocument	—	Документ, входящий в операцию, либо обрабатываемый самостоятельно.
Document Description	—	"Сниппет" документа, сформированный сервером DSS на основе шаблона для сниппета и содержания документа.
Document Preview	—	Визуализированный в человеко-читаемую форму документ, сформированный сервером DSS на основе шаблона для визуализации и содержания документа
Document RawPDF	—	"Сырое" содержание документа (например, текстового файла), преобразованное в формат PDF.
DSSCertificate	—	Сертификат, привязанный к ключу подписи в учетной записи на DSS.

1. Начало работы с myDSS SDK

1.1. Включение библиотеки myDSS SDK в приложение

Минимальная версия Android SDK, необходимая для работы библиотеки - API Level 19 (KitKat).

Библиотека myDSS SDK может быть добавлена в проект в качестве зависимости из maven-репозитория.

Пример корневого файла **`build.gradle`** проекта:

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.

buildscript {
    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.5.1'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()

        maven {
            url "https://repo.paycontrol.org/mydss/android/maven"
            credentials {
                username = "[Можно получить в safetech]"
                password = "[Можно получить в safetech]"
            }
        }
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Добавление зависимости:

```
implementation 'ru.cryptopro.sdk:mydss:<Последняя версия>'
```

1.2. Настройка манифеста приложения

Для работы библиотеки необходимо добавить соответствующие разрешения (для работы с камерой и сетевыми соединениями) в манифест приложения:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

```
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature            android:name="android.hardware.camera.autofocus"
android:required="false"/>
```

Так как библиотека использует собственную реализацию TLS, использующему её приложению необходимо разрешение на использование «открытых» соединений. Для этого свойство **usesCleartextTraffic** приложения должно иметь значение **true**:

```
    <application
        android:usesCleartextTraffic="true"
        ...
```

1.3. Настройка правил минимизации и обфускации

Если в проекте используются инструменты минификации и обфускации **proguard** или **D8**, то для корректной работы библиотеки в файл правил **proguard** (например, **proguard-rules.pro**) необходимо добавить следующие ограничения:

```
    -keep public class ru.cryptopro.mydss.** { *; }
    -dontwarn ru.cryptopro.mydss.**
    -keep public class ru.CryptoPro.** { *; }
    -dontwarn ru.CryptoPro.**
    -keep public class ru.cprocsp.** { *; }
    -dontwarn ru.cprocsp.**
    -keep public class org.ini4j.spi.** { *; }
    -dontwarn org.ini4j.spi.**
    -keep public class java.awt.event.** { *; }
    -dontwarn java.awt.event.**
    -keep public class javax.swing.** { *; }
    -dontwarn javax.swing.**
    -keep public class com.objsys.asn1j.runtime.** { *; }
    -dontwarn com.objsys.asn1j.runtime.**
```

При отсутствии файла **proguard-rules.pro** в проекте его необходимо создать и добавить в конфигурацию конечной сборки в файле **build.gradle**:

```
    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'pr
oguard-rules.pro'
            proguardFiles fileTree('proguard').asList().toArray()
        }
        debug {
            minifyEnabled false
            debuggable true
        }
    }
```

2. Инициализация библиотеки при запуске

В начале работы приложения необходимо инициализировать библиотеку при помощи статического метода **`MyDSS.init`**.

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import ru.cryptopro.mydss.sdk.v2.MyDss;

public class MainActivity extends AppCompatActivity {

    static MyDSS myDSS = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        MyDss.init( getApplicationContext(),
                    MyDss.DSS_LOG_DEBUG,    // устанавливается уровень ло
гирования библиотеки в LogCat: DSS_LOG_DEBUG или DSS_NO_LOGGING
                    new DSSInitCallback()
        {
            @Override
            public void error(DSSError dssError) {
                // Инициализация прошла неудачно. Библиотекой пользоватьс
я нельзя
                Log.e("ERROR", dssError.getMessage());
            }

            @Override
            public void success(MyDss myDssInstance) {
                // Библиотека инициализирована успешно. Необходимо сохран
ить созданный экземпляр myDssInstance для дальнейшего использования
                myDSS = myDssInstance;
            }
        });
    }
}
```

Кроме того, для выполнения защищенных сетевых запросов используется корневой сертификат, до которого строится цепочка проверки TLS-сертификата сервера.

Корневой сертификат должен размещаться в ресурсах приложения.

Приложение может использовать два корневых сертификата: для разработки и для конечного приложения.

Сертификат для разработки должен быть доступен в приложении как ресурс с идентификатором **`R.raw.development_root_cert`** (для этого, например, сохраните сертификат под именем **`res/raw/development_root_cert.crt`**).

Сертификат для конечного приложения должен быть доступен в приложении как ресурс с идентификатором **`R.raw.production_root_cert`** (для этого, например, сохраните сертификат под именем **`res/raw/production_root_cert.crt`**).

Сертификаты должны быть сохранены в формате **`PEM`**.

По умолчанию используется корневой сертификат для разработки.

В release-сборке приложения для установки типа корневого сертификата для конечного приложения вызовите следующий метод сразу после инициализации myDSS SDK:

```
dss.setRootCertificateType(MyDss.RootCertificateType.Production);
```

При необходимости можно явно указать на использование сертификата для разработки:

```
dss.setRootCertificateType(MyDss.RootCertificateType.Development);
```

3. Использование myDSS SDK. Классы и функции

myDSS SDK предоставляет следующие основные классы для работы:

- **myDSS**: Основной класс для работы с MyDss SDK (см. раздел 3.1).
- **DSSUsersManager**: Создаёт и управляет учётными записями пользователя на устройстве (см. раздел 3.2).
- **DSSOperationsManager**: Получает информацию и подписывает операции и документы (см. раздел 3.3).
- **DSSCertificatesManager**: Управляет сертификатами пользователей (см. раздел 3.4).
- **DSSDevicesManager**: Управляет устройствами, привязанными к учётной записи (см. раздел 3.5).
- **DSSPolicyManager**: Получает информацию о сервере DSS (см. раздел 3.6).

3.1. Класс myDSS

3.1.1. Метод init

Инициализация библиотеки. Работа с myDSS SDK начинается с вызова данной функции.

Инициализация включает в себя следующее.

- Синхронизацию внутренних часов библиотеки.
- Инициализацию криптографических функций.
- Установку внешнего вида пользовательского интерфейса по умолчанию.
- Определение наличия прав суперпользователя (root) в устройстве.
- Определение наличия средств антивирусной защиты.
- Определение списка потенциально опасных приложений.

```
public static void init(Context context,
                        int logLevel,
                        DSSInitCallback callback)
```

Параметры:

- **context** — Контекст приложения. Используется для вызова стандартных функций Android SDK, требующих передачи контекста приложения в списке аргументов.
- **logLevel** — Уровень логирования. Используйте **DSS_LOG_DEBUG**, чтобы разрешить логирование и **DSS_NO_LOGGING**, чтобы предотвратить запись в логи любых сообщений от myDSS SDK.
- **callback** — Метод обратного вызова для получения результатов инициализации. При успешной инициализации возвращается экземпляр инициализированной библиотеки, который должен быть сохранён для дальнейшего использования в приложении. Библиотека не может быть инициализирована дважды.

3.1.2. Метод initRNG

Инициализирует генератор случайных чисел, используемый в CryptoPro CSP. Инициализация происходит при помощи биологической последовательности, которая строится при помощи серии касаний экрана пользователем.

```
public boolean initRNG()
```

Возвращает логическое значение **true** при успешной инициализации. Логическое значение **false**, если инициализация прошла неуспешно.

3.1.3. Метод setLogLevel

Установка уровня логирования.

```
public void setLogLevel(int logLevel)
```

Параметры:

- **logLevel** — Уровень логирования. **DSS_LOG_DEBUG** позволяет выводить сообщения об ошибках и информационные сообщения в логи, в то время как **DSS_NO_LOGGING** предотвращает myDSS SDK от вывода каких-либо сообщений в лог.

3.1.4. Метод setRootCertificateType

Определяет тип корневого сертификата, который используется в цепочке проверки сетевых запросов.

```
public void setRootCertificateType(myDSS.RootCertificateType
rootCertificateType)
```

Параметры:

- **rootCertificateType** — тип сертификата.

Предполагается, что **myDSS.RootCertificateType.Development** используется на этапе разработки и тестирования приложений, в то время как **myDSS.RootCertificateType.Production** применяется в приложениях для конечного пользователя. По умолчанию в myDSS SDK используется **myDSS.RootCertificateType.Development**.

Корневой сертификат "для разработки" должен быть доступен как ресурс приложения, идентифицируемый **R.raw.development_root_cert**, в то время как сертификат "для конечного пользователя" должен быть доступен через идентификатор **R.raw.production_root_cert**.

3.1.5. Метод destroy

Завершает использование библиотеки. Попытка использовать какие-либо методы после вызова данного метода ведёт к неопределённому поведению.

```
public void destroy()
```

3.1.6. Метод getVersion

Возвращает версию myDSS SDK.

```
public java.lang.String getVersion()
```

3.1.7. Метод analyzeQR

Определяет содержание заранее раскодированного QR-кода в виде строки.

Возвращает объект, представляющий информацию о типе QR-кода или **null**, если тип установить не удалось.

```
public static DSSQRCode analyzeQR(java.lang.String qrValue)
```

Параметры:

- **qrValue** — Значение QR-кода в виде строки.

3.1.8. Метод activate

Активация QR-кода, если свойство **qrCodeKinit.isActivated** не равно **true**.

```
public static void activate(DSSQRCodeKinit qrCodeKinit,
                            java.lang.String code,
                            DSSQRCodeCallback callback)
```

Параметры:

- **qrCodeKinit** — Объект QR-кода для активации
- **code** — Код активации.
- **callback** — При успешном выполнении возвращает активированный объект DSSQRCodeKinit. Иначе возвращает ошибку.

3.2. Класс DSSUsersManager

3.2.1. Метод acceptAccountChanges

Подтверждение изменений в аккаунте пользователя на DSS.

```
public static void acceptAccountChanges(DSSUser user,
                                        DSSUserCallback callback)
```

Выполнение возможно только при статусе учетной записи на DSS **notVerified**.

Перед выполнением необходимо выполнить **DSSUsersManager.updateStatus** для получения актуального статуса учетной записи пользователя.

При необходимости для подтверждения использовать QR-код **(user.qrVerificationRequired == true)** - запускает сканер QR-кода.

Учетная запись пользователя на DSS переводится в статус **Active**.

Параметры:

- **user** — Объект пользователя.
- **callback** — В случае успеха возвращает созданный объект пользователя **DSSUser**. Иначе возвращает ошибку.

3.2.2. Метод changePassword

Изменение пароля на вектор аутентификации.

```
public static void changePassword(DSSUser user,
                                  boolean requirePassword,
                                  DSSUserCallback callback)
```

Метод запускает последовательность экранов SDK, выполняющих следующее:

- запрос текущего пароля / отпечатка (при необходимости);
- запрос нового пароля / отпечатка с проверкой парольной политики;
- пересохранение объекта в хранилище.

После выполнения **isReadyToSign = true**.

Параметры:

- **user** — Объект пользователя, чей «вектор аутентификации» будет перешифрован.
- **requirePassword** — Требуется ли ввод пароля.
- **callback** — При успешном выполнении возвращает объект пользователя **DSSUser**.

3.2.3. Метод checkApprovalStatus

Проверка статуса запроса на добавление устройства к учетной записи DSS.

```
public static void checkApprovalStatus(DSSUser dssUnapprovedUser,
                                        DSSUserCallback callback)
```

Параметры:

- **dssUnapprovedUser** — Пользователь, чей статус нужно проверить.
- **callback** — Коллбэк метода.

3.2.4. Метод createDSSUser

Создание неподтвержденной учетной записи в КриптоПро DSS с получением вектора аутентификации к ней.

```
public static void createDSSUser(java.lang.String externalId,
                                  java.lang.String alias,
                                  java.lang.String name,
                                  java.lang.String serviceUrl,
                                  java.lang.String deviceName,
                                  PushNotificationData pushData,
                                  boolean requirePassword,
                                  DSSUserCallback callback)
```

Метод запускает последовательность экранов SDK, выполняющих следующее:

- индикацию состояния взаимодействия с DSS;
- запрос пароля / отпечатка с проверкой парольной политики:
  - если **((passPolicy == 0) И (false == requirePassword))**, то используется вариант "БЕЗ ПАРОЛЯ"
  - в противном случае используется вариант "С ПАРОЛЕМ";
  - если **(0 == KEYFLAG_DENY_STORE_WITH_OS_PROTECTION)**, то введенный пароль предлагается сохранить "ПОД ОТПЕЧАТОК";
- регистрацию ключа на сервере (отпечаток устройства, PUSH-идентификатор);
- сохранение объекта в хранилище.

После выполнения **isReadyToSign = false**, созданная учетная запись имеет статус **Installed**.

Параметры:

- **externalId** — Идентификатор сущности вызывающего приложения, к которому будет «привязан» объект.
- **alias** — Человекочитаемый идентификатор устройства.
- **name** — Имя ключа.
- **serviceUrl** — URL для взаимодействия с myDSS SDK.
- **deviceName** — Читаемое название устройства.

- **pushData** — Данные для получения PUSH-уведомлений.
- **requirePassword** — Требуется ли ввод пароля.
- **callback** — При успешном выполнении возвращает созданный объект пользователя **DSSUser**. Иначе возвращает ошибку.

3.2.5. Метод createDSSUserWithApproval

Создание запроса на добавление устройства к учетной записи DSS.

```
public static void createDSSUserWithApproval(java.lang.String
externalId,
                                              java.lang.String alias,
                                              java.lang.String serviceUrl,
                                              java.lang.String deviceName,
                                              PushNotificationData pushData,
                                              java.lang.String uid,
                                              java.lang.String name,
                                              boolean requirePassword,
                                              DSSUserCallback callback)
```

Создаёт нового неподтверждённого пользователя **(user.state = .notConfirmed)** и сохраняет его в хранилище.

Запускает процесс добавления устройства, где:

- в ответ на данный запрос DSS формирует «вектор аутентификации» и передаёт его на данное устройство;
- вектор имеет статус «неподтвержденный».

Подтверждение с другого устройства выполняется через **DSSDevicesManager.listDevices ... DSSDevicesManager.confirm / reject.**

Параметры:

- **externalId** — Идентификатор сущности вызывающего приложения, к которому будет «привязан» объект.
- **alias** — Человекочитаемый идентификатор устройства.
- **serviceUrl** — URL для взаимодействия с myDSS.
- **deviceName** — Читаемое название устройства.
- **pushData** — Данные для получения PUSH-уведомлений.
- **uid** — Токен (адрес) устройства в Push-сети.
- **name** — Имя ключа.
- **requirePassword** — Требуется ли ввод пароля.
- **callback** — При успешном выполнении возвращает созданный объект пользователя **DSSUnapprovedUser**. Иначе возвращает ошибку.

3.2.6. Метод createDSSUserWithInitQR

Создание неподтвержденной учетной записи в КриптоПро DSS с получением вектора аутентификации к ней с использованием QR-кода.

```
public static void createDSSUserWithInitQR(java.lang.String externalId,
                                            java.lang.String alias,
                                            java.lang.String name,
                                            java.lang.String serviceUrl,
                                            java.lang.String deviceName,
                                            PushNotificationData pushData,
```

```
                                                boolean requirePassword,
                                                DSSUserCallback callback)
```

Для запроса используется **Kinit**, полученный из QR-кода.

Метод запускает последовательность экранов SDK, выполняющих следующее:

- индикацию состояния взаимодействия с DSS;
- сканирование QR-кода;
- запрос кода активации (при необходимости);
- запрос пароля / отпечатка с проверкой парольной политики:
  - если **((passPolicy == 0) И (false == requirePassword))**, то использовать вариант "БЕЗ ПАРОЛЯ"
  - в противном случае использовать вариант "С ПАРОЛЕМ";
  - если **(0 == KEYFLAG_DENY_STORE_WITH_OS_PROTECTION)**, то введенный пароль предлагать сохранить "ПОД ОТПЕЧАТОК";
- регистрацию ключа на сервере (отпечаток устройства, PUSH-идентификатор);
- сохранение объекта в хранилище.

После выполнения **isReadyToSign = false**, созданная учетная запись имеет статус **Installed**.

Параметры:

- **externalId** — Идентификатор сущности вызывающего приложения, к которому будет «привязан» объект.
- **alias** — Человекочитаемый идентификатор устройства.
- **name** — Имя ключа.
- **serviceUrl** — URL для взаимодействия с myDSS SDK.
- **deviceName** — Читаемое название устройства.
- **pushData** — Данные для получения PUSH-уведомлений.
- **requirePassword** — Требуется ли ввод пароля.
- **callback** — При успешном выполнении возвращает созданный объект пользователя **DSSUser**. Иначе возвращает ошибку.

3.2.7. Метод delete

Удаление пользователя из хранилища.

```
public static void delete(DSSUser user)
```

Параметры:

- user — пользователь, которого необходимо удалить.

3.2.8. Метод getOperationsHistory

Получение истории операций пользователя на сервисе.

```
public static void getOperationsHistory(DSSUser user,
                                        int count,
                                        int bookmark,
                                        int[] operationCodes,
                                        DSSOperationsHistoryCallback callback)
```

Параметры:

- **user** — Пользователь, чью историю операций нужно показать.
- **count** — Количество операций для показа.
- **bookmark** — Идентификатор записи, относительно которой осуществляется поиск.
- **operationsCodes** — Коды операций для отображения.
- **callback** — Возвращает историю операций или ошибку.

3.2.9. Метод listStorage

Перечисление доступных объектов **DSSUser**. Для каждого **объекта isReadyToSign = false.**

```
public static java.util.List<DSSUser> listStorage()
```

3.2.10. Метод rename

Переименование пользователя в хранилище.

```
public static DSSError rename(DSSUser user,
                              java.lang.String newName)
```

Параметры:

- user — пользователь, чьё имя нужно переименовать;
- newName — новое имя пользователя.

3.2.11. Метод revoke

Отзыв вектора аутентификации на сервере DSS. Делает вектор недействительным.

```
public static void revoke(DSSUser user,
                          DSSUserCallback callback)
```

Параметры:

- **user** — Пользователь, для которого отзывается вектор аутентификации.
- **callback** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

3.2.12. Метод submitPassword

Предъявление пароля на "вектор аутентификации".

```
public static void submitPassword(DSSUser user,
                                  DSSUserCallback callback)
```

Метод запускает последовательность экранов SDK, выполняющих следующее:

- запрос пароля / отпечатка;
- расшифрование «вектора аутентификации»;
- установку **isReadyToSign = true**.

Параметры:

- **user** — Пользователь, чей «вектор аутентификации» будет расшифрован.
- **callback** — При успешном выполнении возвращает объект пользователя **DSSUser**. Иначе возвращает ошибку.

3.2.13. Метод updateStatus

Обновление статуса **DSSUser** информацией с сервера.

```
public static void updateStatus(DSSUser user,
                                DSSUserCallback callback)
```

Параметры:

- **user** — Пользователь для обновления.
- **callback** — В случае успеха вернет экземпляр обновлённого пользователя **DSSUser**. Иначе возвращает ошибку.

3.3. Класс DSSOperationsManager

3.3.1. Метод confirmOperation — Online

Подтверждение операции.

```
public static void confirmOperation(DSSUser user,
                                    DSSOperation operation,
                                    DSSOperationsManager.SignMode signMode,
                                    DSSApproveRequestNetworkCallback callback)
```

Запускает UI для подписи документов по выбранной операции.

isReadyToSign должен быть **true**.

Параметры:

- **operation** — Операция для подтверждения.
- **user** — Пользователь, чью операцию нужно подтвердить.
- **signMode** — режим подписи (при режиме **Offline** запрос на сервер DSS не отправляется).
- **callback** — При успешном выполнении возвращает **DSSApproveRequestNetworkCallback**. Иначе возвращает ошибку.

3.3.2. Метод confirmOperation — готовый запрос на подтверждение

Подтверждение операции путем отправки заранее подготовленного запроса на подтверждение.

```
public static void confirmOperation(DSSUser user,
                                    DSSOperationsManager.ApproveRequest
approveRequest,
                                    DSSNetworkCallback callback)
```

Параметры:

- **user** — Пользователь, чью операцию нужно подтвердить.
- **approveRequest** — Запрос на подтверждение.
- **callback** — Ничего не возвращает в случае успеха. Иначе возвращает ошибку.

3.3.3. Метод getOperationsList

Получение списка операций с сервера DSS.

```
public static void getOperationsList(DSSUser user,
```

```
                                        DSSOperationsManager.OperationType
operationType,
                                        java.lang.String operationId,
                                        DSSOperationsNetworkCallback
callback)
```

Параметры:

- **user** — Пользователь для которого нужно вывести список операций.
- **operationType** — Тип операции (опционально).
- **operationId** — Идентификатор операции (опционально).
- **callback** — При успешном выполнении возвращает список операций **DSSOperationsNetworkCallback**. Иначе возвращает ошибку.

3.3.4. Метод signDocuments — Online

Подпись документов.

```
public static void signDocuments(DSSUser user,
                      java.util.ArrayList<java.lang.String> documentIds,
                      DSSOperationsManager.SignParams params,
                      DSSSignResultNetworkCallback callback)
```

Запускает UI для подписи выбранных документов.

isReadyToSign должен быть **true**.

Параметры:

- **documentsIDs** — Список идентификаторов документов для подписи.
- **user** — Пользователь, подписывающий документы.
- **params** — Параметры подписания.
- **callback** — В случае успеха возвращает результаты подписания **DSSSignResultNetworkCallback**. Иначе возвращает ошибку.

3.3.5. Метод signDocuments — готовый запрос на подпись

Подпись документов путем отправки заранее подготовленного запроса на подпись.

```
public static void signDocuments(DSSUser user,
                      DSSOperationsManager.ApproveRequest approveRequest,
                      DSSSignResultNetworkCallback callback)
```

Параметры:

- **user** — Пользователь, подписывающий документы.
- **approveRequest** — запрос на подписание.
- **callback** — При успешном выполнении не возвращает ничего. Иначе возвращает ошибку.

3.3.6. Метод signDocumentsOffline — Offline

Формирование запроса на подпись без отправки на сервер DSS.

```
public static void signDocumentsOffline(DSSUser user,

java.util.ArrayList<java.lang.String> documentIDs,
                                DSSOperationsManager.SignParams params,
```

```
                                        DSSApproveRequestNetworkCallback callback)
```

Запускает UI для подписания документов по выбранной операции.

isReadyToSign должен быть **true**.

Параметры:

- **documentsIDs** — Идентификаторы документов для подписи.
- **user** — Пользователь, подписывающий документы.
- **params** — Параметры подписания.
- **callback** — В случае успеха возвращает запрос на подписание **DSSApproveRequestNetworkCallback**. Иначе возвращает ошибку.

3.3.7. Метод uploadDocument

Загрузка документа на сервер.

```
public static void uploadDocument(DSSUser user,
                                  java.lang.String title,
                                  java.lang.String snippetTemplate,
                                  java.lang.String previewTemplate,
                                  byte[] content,
                                  DSSDocumentIdNetworkCallback callback)
```

Параметры:

- **user** — объект пользователя.
- **title** — название.
- **snippetTemplate** — шаблон "плашки", HTML.
- **previewTemplate** — шаблон preview, HTML.
- **content** — содержимое.

3.4. Класс DSSCertificatesManager

3.4.1. Метод createCertificate

Создание сертификата/запроса на сертификат.

```
public static void createCertificate(DSSUser user,
                                     int caId,
                                     java.lang.String templateId,

java.util.HashMap<java.lang.String,java.lang.String> DN,
                                     DSSCertificateNetworkCallback callback)
```

Перед выполнением **user.isReadyToSign** должен быть **true**.

Параметры:

- **user** — Пользователь, для которого нужно создать сертификат/запрос.
- **dn** — Различительное имя субъекта в формате [OID компонента имени: Значение компонента имени].
- **templateId** — Идентификатор шаблона сертификата.
- **caId** — Идентификатор обработчика УЦ.
- **callback** — При успешном выполнении возвращает список объектов **DSSCertificate**. Иначе возвращает ошибку.

3.4.2. Метод deleteCertificate

Удаление сертификата/запроса на сертификат.

```
public static void deleteCertificate(DSSUser user,
                                      java.lang.String dssCertificateId,
                                      java.lang.String dssRequestId,
                                      DSSNetworkCallback callback)
```

Параметры:

- **user** — Пользователь-владелец сертификата/запроса на сертификат.
- **dssCertificateId** — Идентификатор удаляемого сертификата.
- **dssRequestId** — Идентификатор удаляемого запроса на сертификат.
- **callback** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

3.4.3. Метод listCertificates

Получение списка сертификатов и запросов на сертификат.

```
public static void listCertificates(DSSUser user,
                                    DSSCertificatesNetworkCallback callback)
```

Параметры:

- **user** — Пользователь, для которого нужно вернуть список.
- **callback** — При успешном выполнении возвращает объект **DSSCertificatesAndRequests**. Иначе возвращает ошибку.

3.4.4. Метод revokeCertificate

Отзыв сертификата.

```
public static void revokeCertificate(DSSUser user,
                        java.lang.String dssCertificateId,
                        DSSCertificatesManager.RevokeReason revokeReason,
                        long revokeDate,
                        DSSNetworkCallback callback)
```

Перед выполнением **user.isReadyToSign** должен быть **true**.

Параметры:

- **user** — Пользователь-владелец сертификата.
- **dssCertificateId** — Идентификатор сертификата.
- **revokeReason** — Причина отзыва сертификата.
- **revokeDate** - Дата отзыва. 0 — если требуется немедленный отзыв сертификата, конкретная дата —если требуется отложенный отзыв сертификата.
- **callback** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

3.4.5. Метод setCertificate

Установка сертификата в DSS.

```
public static void setCertificate(DSSUser user,
                                  java.lang.String certContent,
                                  DSSNetworkCallback callback)
```

Параметры:

- **user** — Пользователь-владелец сертификата/запроса на сертификат.
- **certContent** — Сертификат/запрос на сертификат в формате Base64.
- **callback** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

3.4.6. Метод setCertificateFriendlyName

Установка отображаемого имени сертификата.

```
public static void setCertificateFriendlyName(DSSUser user,
                                               java.lang.String dssCertificateId,
                                               java.lang.String friendlyName,
                                               DSSNetworkCallback callback)
```

Параметры:

- **user** — Пользователь-владелец сертификата.
- **dssCertificateId** — Идентификатор сертификата.
- **friendlyName** — Дружественное имя сертификата.
- **callback** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

3.4.7. Метод setDefaultCertificate

Установка сертификата по умолчанию.

```
public static void setDefaultCertificate(DSSUser user,
                                          java.lang.String dssCertificateId,
                                          DSSNetworkCallback callback)
```

Параметры:

- **user** — Пользователь-владелец сертификата.
- **dssCertificateId** — Идентификатор сертификата.
- **callback** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

3.4.8. Метод suspendCertificate

Приостановка действия сертификата.

```
public static void suspendCertificate(DSSUser user,
                                       java.lang.String dssCertificateId,
                                       long fromTimeStamp,
                                       long toTimeStamp,
                                       DSSNetworkCallback callback)
```

Параметры:

- **user** — Пользователь-владелец сертификата.
- **dssCertificateId** — Идентификатор сертификата.
- **fromTimeStamp** — Время начала приостановки действия сертификата.
- **toTimeStamp** — Время возобновления действия сертификата.
- **callback** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

3.4.9. Метод unSuspendCertificate

Возобновление действия сертификата.

```
public static void unSuspendCertificate(DSSUser user,
```

```
                                        java.lang.String dssCertificateId,
                                        DSSNetworkCallback callback)
```

Параметры:

- **user** — Пользователь-владелец сертификата.
- **dssCertificateId** — Идентификатор сертификата.
- **callback** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

3.5. Класс DSSDevicesManager

3.5.1. Метод approve

Подтверждение добавления ключа на новое устройство.

```
public static void approve(DSSUser user,
                           DSSDevice device,
                           DSSNetworkCallback callback)
```

Параметры:

- **callback** — При успешном выполнении не возвращает ничего. Иначе возвращает ошибку.

3.5.2. Метод listDevices

Получение с DSS списка всех устройств для данного пользователя.

```
public static void listDevices(DSSUser user,
                               DSSDevicesNetworkCallback callback)
```

Параметры:

- **user** — Объект пользователя, для которого будет предоставлен список устройств.
- **callback** — При успешном выполнении возвращает список устройств. Иначе возвращает ошибку.

3.5.3. Метод reject

Отклонение добавления ключа на новое устройство.

```
public static void reject(DSSUser user,
                          DSSDevice device,
                          DSSNetworkCallback callback)
```

Параметры:

- **device** — Объект добавляемого устройства.
- **user** — Объект пользователя добавляемого устройства.
- **callback** — При успешном выполнении не возвращает ничего. Иначе возвращает ошибку.

3.5.4. Метод revoke

Отзыв (удаление) устройства и советующего ему ключа myDSS. Помечает ключ, соответствующий данному устройству, как удаленный. Ключ перестаёт действовать.

```
public static void revoke(DSSUser user,
```

```
                                DSSDevice device,
                                DSSNetworkCallback callback)
```

Параметры:

- **device** — Объект добавляемого устройства.
- **user** — Объект пользователя добавляемого устройства.
- **callback** — При успешном выполнении не возвращает ничего. Иначе возвращает ошибку.

3.6. Класс DSSPolicyManager

3.6.1. Метод getDSSSignParams

Запрос с сервера DSS параметров подписи: список профилей подписи, параметры Удостоверяющих Центров и т.п.

```
public static void getDSSSignParams (DSSUser user,
                                DSSPolicySignServerNetworkCallback callback)
```

Параметры:

- **user** — Объект пользователя.
- **callback** — При успешном выполнении возвращает строку с JSON-ответом. Иначе возвращает ошибку.

3.6.2. Метод getDSSParams

Запрос параметров взаимодействия у сервера DSS.

```
public static void getDSSParams(java.lang.String serviceUrl,
                                java.lang.String rootCert,
                                DSSPolicyNetworkCallback callback)
```

Параметры:

- **serviceUrl** — Адрес сервера.
- **rootCert** — Корневой сертификат.
- **callback** — При успешном выполнении возвращает объект **DSSParams** со списком параметров **DSSPolicyNetworkCallback**. Иначе возвращает ошибку.