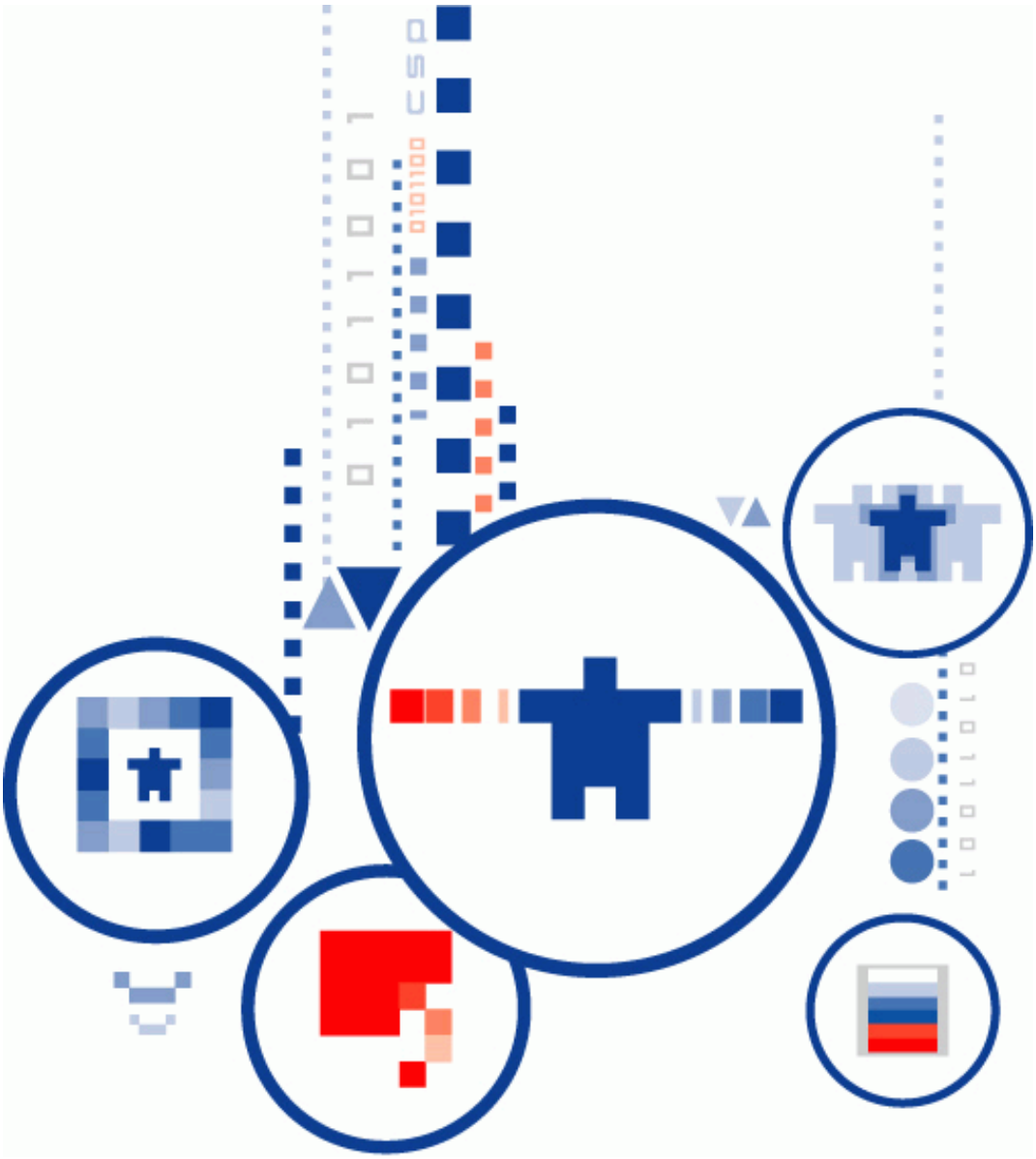

Сервер Электронной Подписи

«КриптоПро DSS»

Компонент ПАКМ «КриптоПро HSM»

myDSS SDK. Руководство разработчика. iOS

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ 2
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 4
1. Начало работы с myDSS SDK 5
1.1. Установка myDSS SDK 5
1.2. Сертификаты 5
1.3. Инициализация myDSS SDK 5
2. Использование myDSS SDK. Классы и функции 6
2.1. Класс myDSSSDK 6
2.1.1. Метод version 6
2.1.2. Метод setLogLevel 6
2.1.3. Метод initialize 6
2.1.4. Метод setRootCertificateType 6
2.1.5. Метод analyzeQR 7
2.1.6. Метод activate 7
2.1.7. Метод initRNG 7
2.2. Класс DSSUsersManager 7
2.2.1. Метод users 7
2.2.2. Метод rename 8
2.2.3. Метод delete 8
2.2.4. Метод updateStatus 8
2.2.5. Метод createDSSUserWithInitQR 8
2.2.6. Метод createDSSUser 9
2.2.7. Метод createDSSUserWithApproval 10
2.2.8. Метод acceptAccountChanges 11
2.2.9. Метод checkApprovalStatus 11
2.2.10. Метод submitPassword 11
2.2.11. Метод changePassword 12
2.2.12. Метод revoke 12
2.2.13. Метод getOperationsHistory 12
2.3. Класс DSSOperationsManager 13
2.3.1. Метод getOperationsList 13
2.3.2. Метод confirmOperation — Online 13
2.3.3. Метод confirmOperation — готовый запрос на подтверждение 14
2.3.4. Метод uploadDocument 14
2.3.5. Метод signDocuments – Online 14
2.3.6. Метод signDocuments – готовый запрос на подписание 15
2.3.7. Метод signDocumentsOffline 15
2.4. Класс DSSCertificatesManager 15
2.4.1. Метод createCertificate 15
2.4.2. Метод listCertificates 16
2.4.3. Метод deleteCertificate 16

2.4.4. Метод setCertificate 16
2.4.5. Метод setCertificateFriendlyName 17
2.4.6. Метод setDefaultCertificate 17
2.4.7. Метод revokeCertificate 17
2.4.8. Метод suspendCertificate 18
2.4.9. Метод unSuspendCertificate 18
2.5. Класс DSSDevicesManager 18
2.5.1. Метод listDevices 18
2.5.2. Метод revoke 19
2.5.3. Метод approve..... 19
2.5.4. Метод reject 19
2.6. Класс DSSPolicyManager 20
2.6.1. Метод getDSSParams 20
2.6.2. Метод getDSSSignServerParams..... 20

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. Установка myDSS SDK

Для работы с библиотекой необходимы Xcode версии 11 и новее. Установка состоит из следующих этапов:

- Скачайте и распакуйте архив из репозитория.
- Добавьте фреймворк **myDSSSDK.xcframework** в основной проект в «**Project Settins -> [Target Name] -> General**» в секции **Frameworks, Libraries** and **Embedded Content**.
- Дополнительные файлы (**config.ini** и **RndmBioViewControllerIPhone.xib**) добавьте в проект и прикрепите к приложению в «**Project Settins -> [Target Name] -> Build Phases**» в секции **Copy Bundle Resources**.

1.2. Сертификаты

Для взаимодействия с серверами в ресурсы приложения необходимо поместить два корневых сертификата (Формат **PEM**).

- **development_root_cert.crt** // Для взаимодействия с тестовым сервером
- **production_root_cert.crt** // Для взаимодействия с основным сервером

Одновременно может быть использован только один сертификат. Он указывается методом **myDSSSDK.setRootCertificateType(_ type: RootCertificateType)**:

```
        .setRootCertificateType .
// или
        .setRootCertificateType .
```

1.3. Инициализация myDSS SDK

myDSS SDK предоставляет возможность инициализировать библиотеку, выставить уровень логирования и указать используемый тип корневого сертификата. Это необходимо сделать перед вызовом внутренних функций myDSS SDK.

```
import UIKit
import
@UIApplicationMain
class AppDelegate  UIResponder  UIApplicationDelegate

    func         application                                UIApplication
                                                 UIApplication.LaunchOptionsKey
Any    -> Bool

        // Установка уровня логирования и типа сертификата
        #if DEBUG
                .setLogLevel  .       .
                .setRootCertificateType .
        #else
                .setRootCertificateType .
        #endif

        // Инициализация библиотеки
                .initialize
        ...
```

2. Использование myDSS SDK. Классы и функции

myDSS SDK предоставляет следующие основные классы для работы:

- **myDSSSDK**: Основной класс для работы с MyDss SDK (см. раздел 2.1).
- **DSSUsersManager**: Создаёт и управляет учётными записями пользователя на устройстве (см. раздел 2.2).
- **DSSOperationsManager**: Получает информацию и подписывает операции и документы (см. раздел 2.3).
- **DSSCertificatesManager**: Управляет сертификатами пользователей (см. раздел 2.4).
- **DSSDevicesManager**: Управляет устройствами, привязанными к учётной записи (см. раздел 2.5).
- **DSSPolicyManager**: Получает информацию о сервере DSS (см. раздел 2.6).

2.1. Класс myDSSSDK

2.1.1. Метод version

Текущая версия библиотеки.

```
public static var version: String
```

2.1.2. Метод setLogLevel

Установка уровня логирования.

```
public static func setLogLevel(_ logLevel: LogLevels)
```

Параметры:

- **logLevel** — Требуемый уровень логирования.

2.1.3. Метод initialize

Инициализация библиотеки. Инициализация включает в себя:

- синхронизацию внутренних часов библиотеки;
- инициализацию криптографических функций;
- установку внешнего вида пользовательского интерфейса по умолчанию.

```
public static func initialize()
```

2.1.4. Метод setRootCertificateType

Устанавливает тип корневого сертификата для взаимодействия с сервером.

Для разработки и тестового режима рекомендуется использовать тип **.development**, для полноценной версии — **.production.**

```
public static func setRootCertificateType(
    _ type: RootCertificateType)
```

Параметры:

- **type** — Тип сертификата.

2.1.5. Метод analyzeQR

Определение содержания QR-кода. После анализа метод возвращает объект с содержимым QR-кода.

```
public static func analyzeQR(
    _ source: String) throws -> DSSQRCode
```

Параметры:

- **source** — Содержимое QR-кода в формате JSON.

Анализ QR-кода и создание объекта требуемого типа. После анализа метод возвращает объект QR-кода требуемого типа или возвращает ошибку, если вводные данные не подходят.

```
public static func analyzeQR<T: DSSQRCode>(
    _ source: String) throws -> T
```

Параметры:

- **source** — Содержимое QR-кода в формате JSON.

2.1.6. Метод activate

Активация QR-кода типа **DSSQRCodeKinit**. Возвращает экземпляр QR-кода с активированной ключевой информацией.

```
public static func activate(
    qrCodeKinit: DSSQRCodeKinit,
    code: String) throws -> DSSQRCodeKinit
```

Параметры:

- **code** — Код активации.

2.1.7. Метод initRNG

Открывает окно ДСЧ.

```
public static func initRNG()
```

2.2. Класс DSSUsersManager

2.2.1. Метод users

Перечисление доступных объектов **DSSUser** в хранилище.

```
public static var users: [DSSUser] { get }
```

2.2.2. Метод rename

Переименование пользователя в хранилище происходит следующим образом. После переименования переданный экземпляр пользователя (**DSSUser**) станет неактуальным. Обновлённый экземпляр можно получить, используя **DSSUsersManager.users**.

```
public static func rename(
    user: DSSUser,
    newName: String)
```

Параметры:

- **user** — Пользователь, чьё имя нужно переименовать.
- **newName** — Новое имя пользователя.

2.2.3. Метод delete

Удаление пользователя из хранилища.

```
public static func delete(
    user: DSSUser)
```

Параметры:

- **user** — Пользователь, которого необходимо удалить.

2.2.4. Метод updateStatus

Обновление статуса **DSSUser** информацией с сервера.

```
public static func updateStatus(
    user: DSSUser,
    _ callback: @escaping (_ result: Result<DSSUser, Error>) -> Void)
```

Параметры:

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

2.2.5. Метод createDSSUserWithInitQR

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

```
public static func createDSSUserWithInitQR(
    serviceURL: URL,
    name: String,
    pushNotificationsData: PushNotificationsData,
    deviceName: String,
    externalId: String?,
    alias: String?,
    requirePassword: Bool,
    callback: @escaping (_ result: Result<DSSUser, Error>) -> Void)
```

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:

- **serviceURL** — Адрес сервера.
- **name** — Имя для сохранения учетной записи.
- **pushNotificationsData** — Данные для отправки PUSH-уведомлений.
- **deviceName** — Отображаемое дружественное имя устройства.
- **externalId** — Внешний идентификатор.
- **alias** — Уникальный человекочитаемый идентификатор, используемый для подтверждения владения МУ пользователем. Если данный параметр отсутствует в запросе, то его значение будет создано автоматически на стороне сервера.
- **requirePassword** — Нужно ли установить пароль.
- **callback** — Замыкание возвращает результат.
- **result** — В случае успеха возвращает экземпляр неподтверждённого пользователя. Иначе возвращает ошибку.

2.2.6. Метод createDSSUser

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

```
public static func createDSSUser(
    serviceURL: URL,
    name: String,
    pushNotificationsData: PushNotificationsData,
    deviceName: String,
    externalId: String?,
    alias: String?,
    requirePassword: Bool,
    callback: @escaping (_ result: Result<DSSUser, Error>) -> Void)
```

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:

- **`serviceURL`** — Адрес сервера.
- **`name`** — Имя для сохранения учетной записи.
- **`pushNotificationsData`** — Данные для отправки PUSH-уведомлений.
- **`deviceName`** — Отображаемое дружественное имя устройства.
- **`externalId`** — Внешний идентификатор.
- **`alias`** — Уникальный человекочитаемый идентификатор, используемый для подтверждения владения МУ пользователем. Если данный параметр отсутствует в запросе, то его значение будет создано автоматически на стороне сервера.
- **`requirePassword`** — Нужно ли установить пароль.
- **`callback`** — Замыкание возвращает результат.
- **`result`** — В случае успеха возвращает экземпляр неподтверждённого пользователя. Иначе возвращает ошибку.

2.2.7. Метод createDSSUserWithApproval

Создание запроса на добавление нового устройства к учетной записи DSS.

```
public static func createDSSUserWithApproval(
    serviceURL: URL,
    uid: String,
    name: String,
    pushNotificationsData: PushNotificationsData,
    deviceName: String,
    externalId: String?,
    alias: String?,
    requirePassword: Bool,
    callback: @escaping (_ result: Result<DSSUser, Error>) -> Void)
```

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:

- **`serviceURL`** — Адрес сервера.
- **`uid`** — Идентификатор пользователя, к которому нужно подключить устройство.
- **`name`** — Имя для сохранения учетной записи.
- **`pushNotificationsData`** — Данные для отправки PUSH-уведомлений.
- **`deviceName`** — Отображаемое дружественное имя устройства.
- **`externalId`** — Внешний идентификатор.

- **alias** — Уникальный человекочитаемый идентификатор, используемый для подтверждения владения МУ пользователем. Если данный параметр отсутствует в запросе, то его значение будет создано автоматически на стороне сервера.
- **requirePassword** — Нужно ли установить пароль.
- **callback** — Замыкание возвращает результат.
- **result** — В случае успеха возвращает экземпляр неподтверждённого пользователя. Иначе возвращает ошибку.

2.2.8. Метод acceptAccountChanges

Подтверждение присоединения мобильного устройства к учётной записи.

```
public static func acceptAccountChanges(
    user: DSSUser,
    callback: @escaping (_ result: Result<DSSUser, Error>) -> Void)
```

Выполнение возможно только при статусе учетной записи в 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** — Пользователь мобильного устройства
- **callback** — Замыкание возвращает результат
- **result** — В случае успеха вернёт экземпляр подтверждённого пользователя. Иначе возвращает ошибку.

2.2.9. Метод checkApprovalStatus

Проверка статуса запроса на добавление устройства к учетной записи DSS.

```
public static func checkApprovalStatus(
    unapprovedUser: DSSUser,
    callback: @escaping (_ result: Result<DSSUser, Error>) -> Void)
```

Параметры:

- **unapprovedUser** — Пользователь, чей статус нужно проверить.
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает пользователя, если он подтверждён. Иначе возвращает ошибку.

2.2.10. Метод submitPassword

Предъявление пароля на вектор аутентификации.

```
public static func submitPassword(
    user: DSSUser,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

Метод запускает последовательность экранов SDK, выполняющих следующее:

- запрос пароля / отпечатка;

- расшифрование вектора аутентификации;
- установку **isReadyToSign = true**.

Параметры:

- **user** — Пользователь, для которого предъявляется пароль.
- **callback** — Замыкание возвращает результат.
- **result** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

2.2.11. Метод changePassword

Изменение пароля на вектор аутентификации.

```
public static func changePassword(
    user: DSSUser,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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** — Пользователь, для которого устанавливается новый пароль.
- **callback** — Замыкание возвращает результат.
- **result** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

2.2.12. Метод revoke

Отзыв вектора аутентификации на сервере DSS. Делает вектор недействительным.

```
public static func revoke(
    user: DSSUser,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

Параметры:

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

2.2.13. Метод getOperationsHistory

Получение истории операций пользователя на сервисе.

```
public static func getOperationsHistory(
    user: DSSUser,
    count: Int = 1,
    bookmark: Int?,
    operationsCodes: [Int] = [],
    callback: @escaping (_ result: Result<DSSOperationsHistory, Error>)
-> Void)
```

Параметры:

- **user** — Пользователь, чью историю операций нужно показать.
- **count** — Количество операций для показа.
- **bookmark** — Идентификатор записи.
- **operationsCodes** — Коды операций для отображения.
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает историю операций или ошибку.

2.3. Класс DSSOperationsManager

2.3.1. Метод getOperationsList

Получение списка операций с сервера DSS.

```
public static func getOperationsList(
    user: DSSUser,
    operationType: String?,
    operationId: String?,
    callback: @escaping (_ result: Result<[DSSOperation], Error>) ->
Void)
```

Параметры:

- **user** — Пользователь для которого нужно вывести список операций.
- **operationType** — Тип операции (опционально).
- **operationId** — Идентификатор операции (опционально).
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает список операций или ошибку.

2.3.2. Метод confirmOperation — Online

Подтверждение операции.

```
public static func confirmOperation(
    operation: DSSOperation,
    user: DSSUser,
    signMode: DSSSignMode,
    callback: @escaping (_ result: Result<DSSApproveRequest, Error>) ->
Void)
```

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: **.online** или **.offline** (при режиме **.offline** запрос на сервер DSS не отправляется).
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает запрос на подтверждение или ошибку.

2.3.3. Метод confirmOperation — готовый запрос на подтверждение

Подтверждение операции путем отправки заранее подготовленного запроса на подтверждение.

```
public static func confirmOperation(
    approveRequest: DSSApproveRequest,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

Параметры:

- **approveRequest** — Запрос на подтверждение.
- **callback** — Замыкание возвращает результат.
- **result** — Ничего не возвращает в случае успеха. Иначе возвращает ошибку.

2.3.4. Метод uploadDocument

Загрузка документа на сервер.

```
public static func uploadDocument(
    documentContent: Data,
    title: String,
    snippetTemplate: String,
    previewTemplate: String,
    user: DSSUser,
    callback: @escaping (_ result: Result<String, Error>) -> Void)
```

Параметры:

- **documentContent** — Содержимое документа.
- **title** — Название документа с расширением. Например, **document.txt**.
- **snippetTemplate** — HTML-шаблон snippet'a документа.
- **previewTemplate** — HTML-шаблон preview документа.
- **user** — Пользователь-владелец документа.
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает идентификатор документа или ошибку.

2.3.5. Метод signDocuments – Online

Подпись документов.

```
public static func signDocuments(
    documentsIDs: [String],
    user: DSSUser,
    signParams: DSSSignParams,
    callback: @escaping (_ result: Result<[DSSSignResult], Error>) ->
Void)
```

Запускает UI для подписи выбранных документов.

isReadyToSign должен быть **true**.

Параметры:

- **documentsIDs** — Идентификаторы документов для подписи.
- **user** — Пользователь, подписывающий документы.
- **signParams** — Параметры подписания.
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает результаты подписания или ошибку.

2.3.6. Метод signDocuments — готовый запрос на подписание

Подпись документов путем отправки заранее подготовленного запроса на подпись.

```
public static func signDocuments(
    approveRequest: DSSApproveRequest,
    callback: @escaping (_ result: Result<[DSSSignResult], Error>) ->
Void)
```

Параметры:

- **approveRequest** — запрос на подписание.
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает результаты подписания или ошибку.

2.3.7. Метод signDocumentsOffline

Формирование запроса на подпись без отправки на сервер DSS.

```
public static func signDocumentsOffline(
    documentsIDs: [String],
    user: DSSUser,
    signParams: DSSSignParams,
    callback: @escaping (_ result: Result<DSSApproveRequest, Error>) ->
Void)
```

Запускает UI для подписания документов по выбранной операции.

isReadyToSign должен быть **true**.

Параметры:

- **documentsIDs** — Идентификаторы документов для подписи.
- **user** — Пользователь, подписывающий документы.
- **signParams** — Параметры подписания.
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает запрос на подписание или ошибку.

2.4. Класс DSSCertificatesManager

2.4.1. Метод createCertificate

Создание сертификата/запроса на сертификат.

```
public static func createCertificate(
    user: DSSUser,
    dn: [String: String],
    templateId: String,
    caId: Int,
    callback: @escaping (_ result: Result<DSSCertificate, Error>) ->
Void)
```

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** — Замыкание возвращает результат.
- **result** — Возвращает созданный запрос на сертификат или ошибку.

2.4.2. Метод listCertificates

Получение списка сертификатов и запросов на сертификат.

```
public static func listCertificates(
    user: DSSUser,
    callback: @escaping (_ result: Result<[DSSCertificate], Error>) ->
Void)
```

Параметры:

- **user** — Пользователь, для которого нужно вернуть список.
- **callback** — Замыкание возвращает результат.
- **result** — Возвращает список или ошибку.

2.4.3. Метод deleteCertificate

Удаление сертификата/запроса на сертификат.

```
public static func deleteCertificate(
    user: DSSUser,
    dssCertificateId: String?,
    dssRequestId: String?,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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

Нужно передать либо **dssCertificateId**, либо **dssRequestId**:

- Если передать **dssCertificateId**, то удалится и сертификат, и связанный с ним запрос.
- Если передать **dssRequestId**, то удалится только запрос.

Параметры:

- **user** — Пользователь-владелец сертификата/запроса на сертификат.
- **dssCertificateId** — Идентификатор удаляемого сертификата. Указывается, если нужно удалить сертификат.
- **dssRequestId** — Идентификатор удаляемого запроса на сертификат. Указывается, если нужно удалить запрос на сертификат
- **callback** — Замыкание возвращает результат.
- **result** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

2.4.4. Метод setCertificate

Установка сертификата в DSS.

```
public static func setCertificate(
    user: DSSUser,
    certContent: String,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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

Параметры:

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

2.4.5. Метод setCertificateFriendlyName

Установка отображаемого имени сертификата.

```
public static func setCertificateFriendlyName(
    user: DSSUser,
    dssCertificateId: String,
    friendlyName: String,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

Параметры:

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

2.4.6. Метод setDefaultCertificate

Установка сертификата по умолчанию.

```
public static func setDefaultCertificate(
    user: DSSUser,
    dssCertificateId: String,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

Параметры:

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

2.4.7. Метод revokeCertificate

Отзыв сертификата.

```
public static func revokeCertificate(
    user: DSSUser,
    dssCertificateId: String,
    reason: CertificateRevokingReason = .unspecified,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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

Параметры:

- **user** — Пользователь-владелец сертификата.
- **dssCertificateId** — Идентификатор сертификата.
- **reason** — Причина отзыва сертификата.

- **callback** — Замыкание возвращает результат.
- **result** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

2.4.8. Метод suspendCertificate

Приостановка действия сертификата.

```
public static func suspendCertificate(
    user: DSSUser,
    dssCertificateId: String,
    fromTimeStamp: TimeInterval,
    toTimeStamp: TimeInterval,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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

Параметры:

- **user** — Пользователь-владелец сертификата.
- **dssCertificateId** — Идентификатор сертификата.
- **fromTimeStamp** — Время начала приостановки действия сертификата. Можно указать 0 — тогда время приостановки будет равно текущему времени.
- **toTimeStamp** — Время возобновления действия сертификата.
- **callback** — Замыкание возвращает результат.
- **result** — В случае успеха не возвращает ничего. Иначе возвращает ошибку.

2.4.9. Метод unSuspendCertificate

Возобновление действия сертификата.

```
public static func unSuspendCertificate(
    user: DSSUser,
    dssCertificateId: String,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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

Параметры:

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

2.5. Класс DSSDevicesManager

2.5.1. Метод listDevices

Получение с DSS списка всех устройств, привязанных к пользователю.

```
public static func listDevices(
    user: DSSUser,
    callback: @escaping (_ result: Result<[DSSDevice], Error>) -> Void)
```

Параметры:

- **user** — Пользователь, к которому привязаны устройства.
- **callback** — Замыкание возвращает результат.

- **result** — При успешном выполнении возвращает список устройств. Иначе возвращает ошибку.

2.5.2. Метод revoke

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

```
public static func revoke(
    device: DSSDevice,
    user: DSSUser,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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

Параметры:

- **device** — Отзываемое устройство.
- **user** — Пользователь к которому привязано устройство.
- **callback** — Замыкание возвращает результат.
- **result** — При успешном выполнении не возвращает ничего. Иначе возвращает ошибку.

2.5.3. Метод approve

Подтверждение добавления ключа на новое устройство.

```
public static func approve(
    device: DSSDevice,
    user: DSSUser,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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

Параметры:

- **device** — Добавляемое устройство.
- **user** — Пользователь к которому привязано устройство.
- **callback** — Замыкание возвращает результат.
- **result** — При успешном выполнении не возвращает ничего. Иначе возвращает ошибку.

2.5.4. Метод reject

Отклонение добавления ключа на новое устройство.

```
public static func reject (
    device: DSSDevice,
    user: DSSUser,
    callback: @escaping (_ result: Result<Void, Error>) -> Void)
```

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

Параметры:

- **device** — Отклоняемое устройство.
- **user** — Пользователь, к которому привязано устройство.
- **callback** — Замыкание возвращает результат.

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

2.6. Класс DSSPolicyManager

2.6.1. Метод getDSSParams

Запрос параметров сервера DSS.

```
public static func getDSSParams(
    serviceURL: URL,
    _ callback: @escaping (_ result: Result<DSSParams, Error>) -> Void)
```

Параметры:

- **serviceURL** — Адрес сервера.
- **callback** — Замыкание возвращает результат.
- **result** — При успешном выполнении возвращает параметры сервера. Иначе возвращает ошибку.

Запрос параметров сервера DSS.

```
public static func getDSSParams(
    user: DSSUser,
    _ callback: @escaping (_ result: Result<DSSParams, Error>) -> Void)
```

Параметры:

user — Пользователь, с сервера которого нужно получить параметры.

callback — Замыкание возвращает результат.

result — При успешном выполнении возвращает параметры сервера. Иначе возвращает ошибку.

2.6.2. Метод getDSSSignServerParams

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

```
public static func getDSSSignServerParams(
    user: DSSUser,
    _ callback: @escaping (_ result: Result<DSSSignServerParams, Error>)
-> Void)
```

Параметры:

- **user** — Пользователь, с сервера которого нужно получить параметры.
- **callback** — Замыкание возвращает результат.
- **result** — При успешном выполнении возвращает параметры сервера. Иначе возвращает ошибку.