

Примеры работы с КриптоПро Архив

Описание всех методов КриптоПро расположено по адресу

<https://archive.cryptopro.ru:32764/swagger> (ссылку открывать в Яндекс.Браузер или Chromium-Gost). Там же их можно опробовать. Адреса методов далее указаны относительно <https://archive.cryptopro.ru:32764>.

Оглавление

Создание контейнера (загрузка документа)	1
Получение информации о существующих контейнерах.....	4
Получение подробной информации о контейнере (проверка наличия документа).	6
Получение статуса контейнера	8
Скачивание подписей в формате CAdES-A	8
Плагин подключения внешней СХД к подсистеме Архив УЦ.....	9
Плагин обработки подписи после усовершенствования	14

Создание контейнера (загрузка документа)

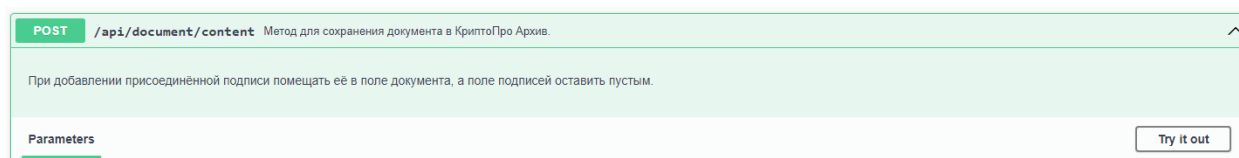
Для создания контейнера необходимо вызвать метод POST /api/document/content.

Тело запроса имеет тип multipart/form-data и состоит из следующих полей:

Имя поля	Описание
Content (обязательно)	Бинарное содержимое подписанного файла. В случае присоединённой подписи — сама подпись.
Signs	Список бинарных содержимых подписей файла. В случае присоединённой подписи поле не передавать. Подписи в кодировке base64 не принимаются.
StorageId (обязательно)	Идентификатор хранилища, в котором необходимо создать контейнер. Список доступных хранилищ с идентификаторами можно получить методом GET /api/user/storages.

SaveType (обязательно)	Тип хранения контейнера. Может быть временным (Temporary) или постоянным (Permanent). В случае временного типа хранения необходимо также передать дату окончания хранения в параметре ArchiveDate .
ContainerName (обязательно)	Имя контейнера.
StructureName	Имя структурного подразделения.
ArchiveDate	Дата окончания хранения. Обязательна в случае, если в параметре SaveType выбран тип хранения временный (Temporary).
IsNeedArchivatorSign	Не используемый флаг, оставленный для обратной совместимости. Не передавать.
UseArchiveCa	Флаг требования использования модуля Архив УЦ для сохранения в нём подписанного документа.
ArchiveCaPluginName	Уникальное имя плагина подсистемы Архив УЦ. Если не указан, документ будет сохранён в базе данных подсистемы Архив УЦ.

Для того, чтобы опробовать метод в Swagger, щёлкните по нему и нажмите **Try it out**:



Пример заполнения формы в Swagger:

POST /api/document/content Метод для сохранения документа в КриптоПро Архив.

При добавлении присоединённой подписи помещать её в поле документа, а поле подписей оставить пустым.

Parameters Cancel Reset

No parameters

Request body multipart/form-data

UseArchiveCa
boolean
Флаг требования использования подсистемы Архив УЦ.
false
 Send empty value

Content * required
string(\$binary)
Содержимое подписанного файла. В случае присоединённой подписи приложить её сюда.
Выберите файл document.txt

Signs
array
Набор подписей документа. В случае присоединённой подписи оставить поле пустым. Подписи в кодировке base64 не принимаются.
Выберите файл x11-detached.sig -
Выберите файл a-detached.sig -
Add string item
 Send empty value

StorageId * required
integer(\$int32)
Идентификатор хранилища, в которое поместить документ.
4

Save Type * required
integer(\$int32)
Temporary

ContainerName * required
string
Имя контейнера (документа).
MyContainer

StructureName
string
Имя структурного подразделения.
StructureName
 Send empty value

ArchiveDate
string(\$date-time)
Дата конца архивации.
21.07.2023
 Send empty value

IsNeedArchivatorSign
boolean
Флаг требования подписи архивариуса.
--
 Send empty value

Execute Clear

Обратите внимание, что в полях **StructureName** и **IsNeedArchivatorSign** не отмечен пункт **Send empty value**. Это означает, что при выполнении запроса Swagger не будет посылать эти поля. В противном случае эти поля были бы отправлены на сервер с пустыми значениями.

Чтобы отправить запрос, нажмите **Execute**.

После выполнения запроса мы можем посмотреть на пример curl-запроса, который был сформирован:

```
curl -X 'POST' \  
  'https://archive.cryptopro.ru:32764/api/document/content' \  
  -H 'accept: text/plain' \  
  -H 'Content-Type: multipart/form-data' \  
  -F 'ContainerName=MyContainer' \  
  -F 'ArchiveDate=21.07.2023' \  
  -F 'StorageId=4' \  
  -F 'Signs=@xlt1-detached.sig;type=application/sig' \  
  -F 'Signs=@a-detached.sig;type=application/sig' \  
  -F 'SaveType=Temporary' \  
  -F 'Content=@document.txt;type=text/plain' \  
  -F 'UseArchiveCa=false'
```

Пример ответа с комментариями:

```
{  
  "data": "bc32d0e9-d085-4236-9e46-ba88d0490a41", // идентификатор созданного контейнера  
  "errorCode": null, // код ошибки  
  "errorDescription": null // текст ошибки  
}
```

В поле data находится идентификатор созданного контейнера. По нему можно получать доступ к информации о контейнере и к подписям контейнера. Этот идентификатор мы будем использовать далее.

Получение информации о существующих контейнерах

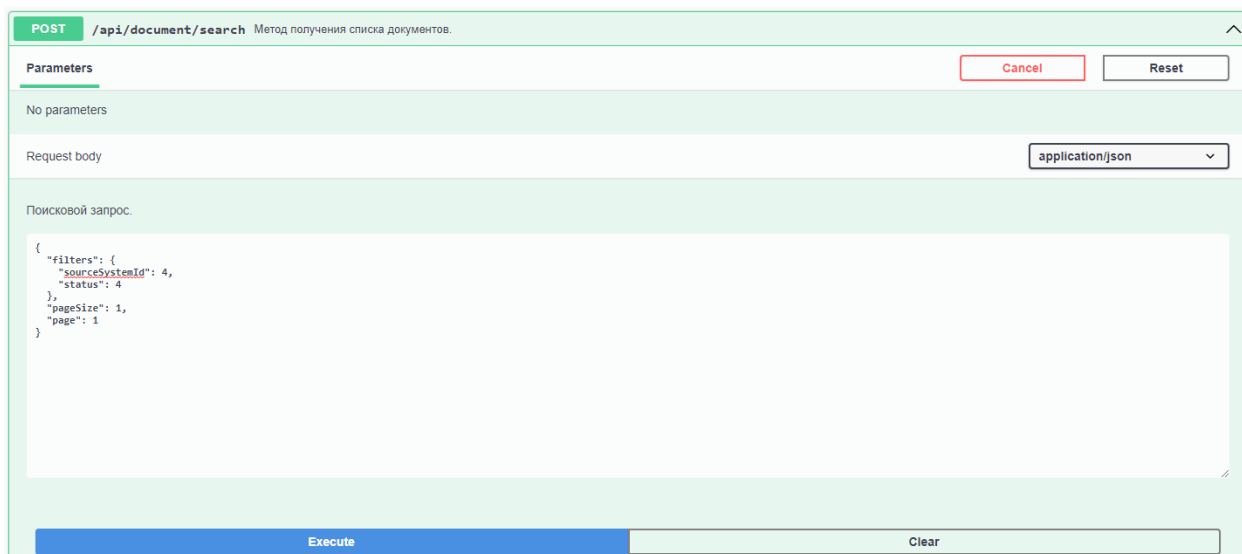
Получить идентификаторы уже существующих контейнеров можно методом POST `/api/document/search`. Тело запроса состоит из полей **filters**, **pageSize** и **page**.

В **filters** указываются фильтры поиска. Если какой-то из фильтров не требуется применять, его не следует указывать в запросе. Если никакие фильтры не требуется применять, поле **filters** можно опустить целиком. Доступные фильтры:

Имя фильтра	Пример	Описание
<code>storageIds</code>	[1, 2, 3]	Список идентификаторов хранилищ, в которых проводить поиск.
<code>name</code>	"MyContainer"	Подстрока имени контейнера.
<code>sourceSystemId</code>	7	Идентификатор пользователя, загрузившего контейнер.
<code>status</code>	4	Статус контейнера.
<code>isArchived</code>	true	Флаг, указывающий на то, что контейнер находится на архивном хранении. Если передать значение false , будут показаны только контейнеры с типом хранения Временный , хранение которых завершено.

`pageSize` (по умолчанию 10) — количество контейнеров, которые нужно вернуть на одной странице. `page` (по умолчанию 1) — номер страницы. Нумерация начинается с единицы. Соответственно, при отправке запроса с пустым телом будут возвращены последние 10 добавленных контейнеров.

Пример поиска контейнеров в статусе **Архивное хранение**, которые загрузил пользователь с идентификатором **4**. Запрашивается первая страница с одним контейнером на странице:



Пример ответа с комментариями:

```
{
```

```

"data": {
  "documents": [ // список контейнеров на запрашиваемой странице
    {
      "id": "bc32d0e9-d085-4236-9e46-ba88d0490a41", // идентификатор контейнера
      "name": "MyContainer", // имя контейнера
      "status": 4, // статус контейнера
      "isArchived": true // флаг, указывающий, что контейнер находится на архивном
хранении
    }
  ],
  "totalRecords": 11 // общее количество найденных контейнеров
},
"errorCode": null,
"errorDescription": null
}

```

Получение подробной информации о контейнере (проверка наличия документа)

Для получения информации о контейнере предназначен метод GET `/api/document/{id}`. Метод возвращает подробную информацию о контейнере. В случае, если контейнер отсутствует в системе, метод вернёт ошибку **404 (Not found)**.

Пример заполнения формы в Swagger:

GET /api/document/{id} Метод для получения подробных сведений о документе.

Parameters

Name	Description
id * required string(\$suid) (path)	Идентификатор документа в системе.

bc32d0e9-d085-4236-9e46-ba88d0490a41

Execute

Cancel

Пример ответа с комментариями:

```

{
  "data": {

```

```

"storage": { // хранилище, в котором находится контейнер
  "id": 4,
  "name": "Техническое сопровождение"
},
"sourceSystem": { // пользователь, создавший контейнер
  "id": 4,
  "name": "Служба технического сопровождения"
},
"hashes": [
  "W0zNKi7Ed44N+/mn0sbZFqzS7RSKdm6CU5SsuA964a0=",
  "W0zNKi7Ed44N+/mn0sbZFqzS7RSKdm6CU5SsuA964a0="
],
"signatures": [ // подписи, хранимые в контейнере
  "<base64signature1>",
  "<base64signature2>"
],
"isToBeSigned": false,
"useArchiveCa": false,
"documentType": "text/plain", // MIME-тип подписанного документа
"saveType": 1, // тип хранения (в данном случае Временный)
"structureName": null, // имя структурного подразделения
"modifiedDate": "2023-05-30T20:00:35.17556+03:00", // дата последнего изменения
"isCreateDate": "2023-05-30T20:00:32.183581+03:00",
"createDate": "2023-05-30T20:00:32.18363+03:00", // дата создания контейнера
"signatureUpdateDate": "2024-01-05T15:50:05+03:00", // дата добавления следующего
архивного штампа к сохранённым подписям
"archiveUntilDate": "2023-07-21T00:00:00+03:00", // дата окончания хранения
"isAvailable": true,
"id": "bc32d0e9-d085-4236-9e46-ba88d0490a41", // идентификатор контейнера
"name": "MyContainer", // имя контейнера
"status": 4, // статус контейнера
"isArchived": true
},
"errorCode": null,
"errorDescription": null
}

```

Получение статуса контейнера

Если необходимо запросить только статус контейнера, используйте метод GET `/api/document/{id}/status`. Вызов этого метода аналогичен вызову, описанному выше. В поле `data` ответа будет возвращено число, соответствующее статусу контейнера. Для получения соответствия цифры статуса его имени используйте метод GET `/api/dictionary/statuses`.

В случае, если контейнер отсутствует в системе, метод вернёт ошибку **404 (Not found)**.

Скачивание подписей в формате CAdES-A

Для скачивания подписей, сохранённых в контейнере, используйте метод GET `/api/document/{id}/sign`. Если вы загрузили подписи, но они не успели усовершенствоваться, метод вернёт те неусовершенствованные подписи, которые вы загрузили. Индикатором того, что подписи были усовершенствованы, служит статус контейнера **Архивное хранение**. Метод возвращает zip-архив с подписями. Тип содержимого ответа — `application/zip`.

После выполнения в Swagger появится ссылка на скачивание (**Download file**):

GET /api/document/{id}/sign Метод скачивания подписей для документа с GUID id.

Parameters

Name	Description
id * required string(\$uuid) (path)	Идентификатор документа.

bc32d0e9-d085-4236-9e46-ba88d0490a41

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'https://archive.cryptopro.ru:32764/api/document/bc32d0e9-d085-4236-9e46-ba88d0490a41/sign' \
-H 'accept: */*'
```

Request URL

```
https://archive.cryptopro.ru:32764/api/document/bc32d0e9-d085-4236-9e46-ba88d0490a41/sign
```

Server response

Code	Details
200	<p>Response body Download file</p> <p>Response headers</p> <pre>content-disposition: attachment; filename=signatures.zip; filename*=UTF-8''signatures.zip content-length: 13458 content-type: application/zip date: Wed, 31 May 2023 09:28:13 GMT server: Kestrel</pre>

Для скачивания нажмите на неё.

Плагин подключения внешней СХД к подсистеме Архив УЦ

В данном разделе описан механизм создания плагина подключения внешней СХД к подсистеме Архив УЦ. В качестве примера рассмотрим процесс написания плагина, сохраняющего подписанные документы на жёсткий диск.

Плагин представляет собой динамическую библиотеку, написанную на языке C#, в которой содержится реализация интерфейса `IDocumentStoragePlugin` из библиотеки `CryptoPro.Archive.Plugins.Design`.

Плагин необходимо расположить в папке `cp-archive/plugins` (она расположена рядом с папками `admin-api`, `client-api` и так далее). По умолчанию это `/opt/cp-archive/plugins` на UNIX и `C:\inetpub\cp-archive\plugins` на Windows. Плагин используется программой `document-consumer` — убедитесь, что она установлена.

Перейдём к примеру написания собственного плагина. Для начала создайте C#-проект и в конфигурационном файле .csproj укажите следующие настройки:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <EnableDynamicLoading>true</EnableDynamicLoading>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="CryptoPro.Archive.Plugins.Design">
      <HintPath>path/to/CryptoPro.Archive.Plugins.Design.dll</HintPath>
      <Private>>false</Private>
      <ExcludeAssets>runtime</ExcludeAssets>
    </Reference>
  </ItemGroup>
</Project>
```

Красным цветом обозначены важные строки. Замените значение HintPath из секции ItemGroup на путь к скачанной библиотеке CryptoPro.Archive.Plugins.Design.dll.

Далее реализуйте интерфейс `IDocumentStoragePlugin`. Ниже приведён пример плагина, сохраняющего документы на жёсткий диск.

```
using CryptoPro.Archive.Plugins.Design.Interfaces;
using CryptoPro.Archive.Plugins.Design.Models;

namespace FilesystemStoragePlugin;

/// <summary>
```

```

/// Пример плагина, позволяющего хранить документы подсистемы Архив УЦ в файловой системе.
/// </summary>
public class FilesystemStoragePlugin : IDocumentStoragePlugin
{
    private string _documentsFolderPath;

    /// <inheritdoc/>
    public void Initialize(Dictionary<string, string> parameters)
    {
        var parametersIgnoreCase = new Dictionary<string, string>(
            parameters, StringComparer.InvariantCultureIgnoreCase);
        _documentsFolderPath = parametersIgnoreCase["DocumentsFolderPath"];
    }

    /// <inheritdoc/>
    public Result<string> Save(Container container)
    {
        var filepath = GenerateFilepath(container.Id);

        try
        {
            File.WriteAllBytes(filepath, container.Document.Raw);
            return filepath;
        }
        catch (Exception exception)
        {
            return Result.Failure<string>(
                $"Unable to save document. Error message: \"{exception.Message}\".");
        }
    }

    /// <inheritdoc/>
    public Result<Document> Get(string path)
    {
        if (!File.Exists(path))

```

```

    {
        return Result.Failure<Document>($"File not found. Path: \"{path}\".");
    }

    try
    {
        var raw = File.ReadAllBytes(path);
        return new Document(raw);
    }
    catch (Exception exception)
    {
        return Result.Failure<Document>(
            $"Unable to get document. Error message: \"{exception.Message}\".");
    }
}

/// <inheritdoc/>
public Result Delete(string path)
{
    if (!File.Exists(path))
    {
        return Result.Failure($"Unable to delete file \"{path}\". File not found.");
    }

    try
    {
        File.Delete(path);
        return Result.Success();
    }
    catch (Exception exception)
    {
        return Result.Failure(
            $"Unable to delete file \"{path}\". Error: \"{exception.Message}\".");
    }
}
}

```

```

/// <summary>Сгенерировать путь к файлу на основе идентификатора контейнера.</summary>
/// <param name="containerId">Идентификатор контейнера.</param>
/// <returns>Путь к файлу.</returns>
private string GenerateFilepath(Guid containerId) =>
    Path.Combine(_documentsFolderPath, containerId.ToString());
}

```

Плагин не должен генерировать исключения. Если во время выполнения какой-либо операции произошла ошибка, используйте класс `Result` для передачи информации об этой ошибке в вызывающую функцию.

Также обратите внимание, что при создании плагина будет вызван конструктор без параметров. Для инициализации плагина используйте функцию `void Initialize(Dictionary<string, string> parameters)`. Параметр `parameters` будет прочитан из секции `Plugins` конфигурационного файла `plugin-config.json` из папки с плагинами `sr-archive/plugins`. Если файл отсутствует, создайте его. Пример содержимого конфигурационного файла для плагина выше:

```

{
  "Plugins": {
    "DocumentStorage": [
      {
        "IsEnabled": true,
        "ReadOnly": false,
        "PluginName": "filesystem_01",
        "AssemblyName": "FilesystemStoragePlugin.dll",
        "Parameters": {
          "DocumentsFolderPath": "/home/username/documents"
        }
      }
    ]
  }
}

```

```
}
```

Секция `Plugins.DocumentStorage` содержит список, что позволяет подключить несколько плагинов одновременно. Параметр `PluginName` содержит имя плагина, которое используется для его идентификации в дальнейшем. Имя должно быть уникальным и непустым.

Выбор плагина при создании контейнера осуществляется при вызове `POST /api/document/content` с помощью указания имени плагина в необязательном параметре `ArchiveCaPluginName` метода.

Плагин обработки подписи после усовершенствования

В данном разделе описан механизм создания плагина обработки подписей после усовершенствования. В качестве примера рассмотрим процесс написания плагина, сохраняющего подписи на жёсткий диск и создающего небольшой отчёт об усовершенствовании для каждого контейнера.

Плагин представляет собой динамическую библиотеку, написанную на языке `C#`, в которой содержится реализация интерфейса `ISanConnectorPlugin` из библиотеки `CryptoPro.Archive.Plugins.Design`.

Плагин необходимо расположить в папке `cp-archive/plugins` (она расположена рядом с папками `admin-api`, `client-api` и так далее). По умолчанию это `/opt/cp-archive/plugins` на UNIX и `C:\inetpub\cp-archive\plugins` на Windows. Плагин используется программой `signature-updater` — убедитесь, что она установлена.

Создайте `C#`-проект и в конфигурационном файле `.csproj` укажите следующие настройки:

```

<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0</TargetFramework>
    <ImplicitUsings>enable</ImplicitUsings>
    <Nullable>enable</Nullable>
    <RootNamespace>FilesystemSanConnectorPlugin</RootNamespace>
    <EnableDynamicLoading>true</EnableDynamicLoading>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="CryptoPro.Archive.Plugins.Design">
      <HintPath>path/to/CryptoPro.Archive.Plugins.Design.dll</HintPath>
      <Private>>false</Private>
      <ExcludeAssets>runtime</ExcludeAssets>
    </Reference>
  </ItemGroup>
</Project>

```

Красным цветом обозначены важные строки. Замените значение HintPath из секции ItemGroup на путь к скачанной библиотеке CryptoPro.Archive.Plugins.Design.dll. Замените значение RootNamespace на имя пространства имён плагина.

Далее реализуйте интерфейс `ISanConnectorPlugin`. Ниже приведён пример плагина, сохраняющего подписи в директорию на жёстком диске и создающего небольшой отчёт о результате усовершенствования.

```

using CryptoPro.Archive.Plugins.Design.Interfaces;
using CryptoPro.Archive.Plugins.Design.Models;

namespace FilesystemSanConnectorPlugin;

/// <summary>
/// Пример плагина, сохраняющего подписи в папку и создающего небольшой отчёт.

```

```

/// </summary>
public class FilesystemSanConnectorPlugin : ISanConnectorPlugin
{
    private string _rootFolder;

    /// <inheritdoc/>
    public void Initialize(Dictionary<string, string> parameters)
    {
        var parametersIgnoreCase = new Dictionary<string, string>(
            parameters, StringComparer.InvariantCultureIgnoreCase);

        if (!parametersIgnoreCase.TryGetValue("folder", out _rootFolder))
        {
            _rootFolder = "";
        };
    }

    /// <inheritdoc/>
    public async Task<Result> ProcessAsync(Container container)
    {
        var containerFolder = GenerateContainerFolderName(container.Id);
        if (!Directory.Exists(containerFolder))
        {
            Directory.CreateDirectory(containerFolder);
        }

        var writeSignaturesResult = await WriteSignaturesAsync(
            containerFolder, container.Signatures);
        if (writeSignaturesResult.IsFailure)
        {
            return writeSignaturesResult;
        }

        return await WriteReportAsync(containerFolder, container);
    }
}

```



```

/// <summary>Записать список подписей в указанную директорию.</summary>
private async Task<Result> WriteSignaturesAsync(
    string folder, IReadOnlyList<Signature> signatures)
{
    for (var i = 0; i < signatures.Count; i++)
    {
        var signature = signatures[i];
        var writeSignatureResult = await WriteSignatureAsync(folder, signature, i);
        if (writeSignatureResult.IsFailure)
        {
            return writeSignatureResult;
        }
    }

    return Result.Success();
}

/// <summary>Записать подпись в указанную директорию.</summary>
private async Task<Result> WriteSignatureAsync(
    string folder, Signature signature, int signatureIndex)
{
    var signaturePath = GenerateSignaturePath(folder, signatureIndex);
    try
    {
        await File.WriteAllBytesAsync(signaturePath, signature.Raw);
    }
    catch (Exception exception)
    {
        return Result.Failure(
            $"Unable to write to file {signaturePath}. Error: {exception.Message}.");
    }

    return Result.Success();
}

```

```

/// <summary>Записать отчёт о контейнере в директорию.</summary>
private async Task<Result> WriteReportAsync(string folder, Container container)
{
    var reportPath = GenerateReportPath(folder);

    var clearFileResult = await DeleteFileContentsAsync(reportPath);
    if (clearFileResult.IsFailure)
    {
        return clearFileResult;
    }

    if (container.Status != ContainerStatus.Archived)
    {
        return await WriteReportFailureAsync(reportPath, container);
    }

    return await WriteReportSuccessAsync(reportPath, container);
}

/// <summary>Записать отчёт об успешном усовершенствовании.</summary>
private async Task<Result> WriteReportSuccessAsync(
    string reportPath, Container container)
{
    for (var i = 0; i < container.Signatures.Count; i++)
    {
        var signature = container.Signatures[i];

        var report = GenerateReport(signature, i);
        try
        {
            await File.AppendAllTextAsync(reportPath, report);
        }
        catch (Exception exception)
        {

```

```

        return Result.Failure(
            $"Unable to write to file {reportPath}. Error: {exception.Message}.");
    }
}

return Result.Success();
}

/// <summary>Записать отчёт о неуспешном усовершенствовании.</summary>
private async Task<Result> WriteReportFailureAsync(
    string reportPath, Container container)
{
    try
    {
        await File.AppendAllTextAsync(
            reportPath, $"Error. Container status: {container.Status}.");
    }
    catch (Exception exception)
    {
        return Result.Failure(
            $"Unable to write to file {reportPath}. Error: {exception.Message}.");
    }

    return Result.Success();
}

/// <summary>Сгенерировать путь к файлу контейнера.</summary>
private string GenerateContainerFolderName(Guid containerId) =>
    Path.Combine(_rootFolder, containerId.ToString());

/// <summary>Сгенерировать путь к файлу подписи.</summary>
private string GenerateSignaturePath(string folderName, int signatureIndex) =>
    Path.Combine(folderName, $"signature-{signatureIndex}.sig");

/// <summary>Сгенерировать путь к отчёту.</summary>

```

```

private string GenerateReportPath(string folderName) =>
    Path.Combine(folderName, @"report.txt");

/// <summary>Сгенерировать отчёт о подписи.</summary>
private string GenerateReport(Signature signature, int signatureIndex) =>
    $"Signature {signatureIndex} expiration date: {signature.Expires}.\n";

/// <summary>Удалить текущее содержимое файла.</summary>
private async Task<Result> DeleteFileContentsAsync(string filepath)
{
    try
    {
        await File.WriteAllTextAsync(filepath, @"");
    }
    catch (Exception exception)
    {
        return Result.Failure(
            $"Unable to clear file {filepath}. Error: {exception.Message}.");
    }

    return Result.Success();
}
}

```

Плагин не должен генерировать исключения. Если во время выполнения какой-либо операции произошла ошибка, используйте класс `Result` для передачи информации об этой ошибке в вызывающую функцию.

Также обратите внимание, что при создании плагина будет вызван конструктор без параметров. Для инициализации плагина используйте функцию `void Initialize(Dictionary<string, string> parameters)`. Параметр `parameters` будет прочитан из секции `Plugins` конфигурационного файла `plugin-config.json` из папки с плагинами `sr-archive/plugins`. Если файл отсутствует, создайте его. Пример содержимого конфигурационного файла для плагина выше:

```
{
  "Plugins": {
    "SanConnector": [
      {
        "IsEnabled": true,
        "PluginName": "SanConnector_01",
        "AssemblyName": "FilesystemSanConnectorPlugin.dll",
        "Parameters": {
          "Folder": "/home/username/containers"
        }
      }
    ]
  }
}
```

Секция `Plugins.SanConnector` содержит список, что позволяет подключить несколько плагинов одновременно. Параметр `PluginName` содержит имя плагина, которое используется для его идентификации в дальнейшем. Имя должно быть уникальным и непустым.

При использовании нескольких плагинов все плагины с полем `IsEnabled` со значением `true` будут вызваны по очереди. Очередь может не соответствовать указанной в списке в конфигурационном файле.