# Прописывание документа на клиенте.

Подпись осуществляется с помощью Browser плагина и криптопровайдера установленного на клиентской машине. Плагин должен содержать реализации работы с различными криптопровайдерами, а конкретная реализация выбирается с помощью настроек. Реализации работы с криптопровайдерами могут взаимодействовать с JCP, SCP, аппаратными ключами, но иметь один интерфейс. Плагин взаимодействует с платформой с помощью web сервисов. На рисунке 1 отображена схема взаимодействия компонент подсистемы ЭП.



Алгоритм работы модуля:

* На странице, содержащей функциональность ЭП размещается плагин. На странице плагин должен быть в единственном экземпляре.
* При своей инициализации страницы с плагином java-script обращается по web-service API к платформе за инициализационными данными. В этих данных присутствует вся информация, необходимая для выбора криптопровайдера, указания корректных криптоалгоритмов, указания типа ЭП итд.
* При нажатие кнопки «Подписать», java-script вызывает функцию web-service и передает туда идентификатор подписываемого объекта.
* Web сервис исходя из типа запрашиваемого объекта и конфигурации модуля формирует контент(ы) для подписи и отправляет его клиенту.
* Java-script отдает плагину контент или хаш (в зависимости от настроек) и плагин выполняет ЭП контента и java-script передает полученную подпись и сертификат обратно на сервер.
* Сервер сохраняет ЭП в хранилище с помощью настраиваемых спринг бинов.
* Java-script формирует сообщение об успешности (ошибке) выполнения подписи.

# Крипто модуль имеет свою xml конфигурацию, которая хранится на сервере и обрабатывается сервисом конфигурации. Конфигурация содержит глобальные настройки, так и настройки для конкретных типов. Глобальные настройки содержат информацию о применяемом крипто провайдере и соответственно имя клиентских и серверных крипто модулей, информацию о криптоалгоритмах, типе ЭП и иную информацию, необходимую для работы плагинов. В глобальных настройках должен может быть раздел, который конфигурирует конкретного криптопровайдера и имеющим свободную структуру, то есть в xsd элемент any.

Глобальные настройки криптопровайдера могут быть только одни в конфигурации и это контролируется валидатором.

Так же в конфигурации настраивается способ получения контента по идентификатору. Подписываемое вложение может быть как файл документа так и XML с атрибутами. Алгоритм получения подписываемого контента задается в конфигурации. Формат xml конфигурации приведен ниже.

<type-crypto-settings name="country\_attachment"

get-content-bean-name="currentAttachmentSignatureDataService"

signature-storage-bean-name="domainObjectStoreSignatureService">

<signature-storage-bean-settings>

<domain-object-store-signature

signature-store-type-name="digital\_signature"

signed-attachment-field-name="signed\_object"

signature-field-name="signature"/>

</signature-storage-bean-settings>

</type-crypto-settings>

Настройка получения подписываемых контентов по идентификатору доменного объекта производится тегом type-crypto-settings. Структура xml:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| Name | Атрибут | Имя типа доменного объекта к которому относится конфигурация |
| get-content-bean-name | Атрибут | Имя спринг бина с интерфейсом SignatureDataService. Бин формирует подписываемый контент |
| signature-storage-bean-name | Атрибут | Имя спринг бина с интерфейсом SignatureStorageService. Бин сохраняет и зачитывает ЭП их хранилища |
| get-content-bean-settings | Элемент | Конфигурация конкретного бина формирования контента для ЭП. Внутри данного тега структура произвольна и определяется самим бином |
| signature-storage-bean-settings | Элемент | Конфигурация конкретного бина хранения ЭП. Внутри данного тега структура произвольна и определяется самим бином |

Разработчики прикладных приложений сами разрабатывают бины для формирования контента для подписи и его сохранения в хранилище исходя из своих задач.

# Проверка ЭП на сервере.

Необходимо реализовать сервис проверки ЭП на сервере. Сервис должен быть выполнен в виде EJB со следующим интерфейсом:

//\*\*

\* Крипто сервис. Серверный компонент для обеспечения работы функций формирования и проверки электронной подписи

\* @author larin

\*

\*/

public interface CryptoService{

public interface Remote extends CryptoService {

}

/\*\*

\* Проверка усовершенствованной ЭП включеннной в документ

\* @param inputStream поток содержащий документ со встроенной электронной подписью

\* @return

\*/

VerifyResult verify(InputStream document);

/\*\*

\* Проверка Усовершенствованной ЭП не включенной в документ

\* @param document поток содержащий документ

\* @param signature электронная подпись

\* @return

\*/

VerifyResult verify(InputStream document, byte[] signature);

/\*\*

\* Проверка стандартной ЭП

\* @param document поток содержащий документ

\* @param signature электронная подпись

\* @param signerSertificate сертификат подписавшено документ в формате DER

\* @return

\*/

VerifyResult verify(InputStream document, byte[] signature, byte[] signerSertificate);

/\*\*

\* Формирование hash переданного документа

\* @param document

\* @return

\*/

byte[] hash(InputStream document);

/\*\*

\* Получение конфигураций из глобальных настроек крипто модуля

\* @return

\*/

CryptoSettingsConfig getCryptoSettingsConfig();

/\*\*

\* Получение контента для ЭП по идентификатору

\* @param id

\* @return

\*/

SignedDataItem getContentForSignature(Id id);

/\*\*

\* Получение идентификаторов связанных доменных объетов контенты которых надо подписать.

\* Например при подписание документа реально подписываются вложения к этому документу.

\* Тоесть при передаче на вход функции идентификатора документа в возвращаемом результате будут идентификаторы вложений к этому документу

\* @param rootId

\* @return

\*/

List<Id> getBatchForSignature(Id rootId);

/\*\*

\* Cохранение электронной подписи в хранилище

\* @param signedResult

\*/

void saveSignedResult(SignedResultItem signedResult);

/\*\*

\* Проверка ЭП вложения (ДО наследника attachment)

\* @param documentId идентификатор документа

\* @return

\*/

List<DocumentVerifyResult> verify(Id documrntId);

}

В качестве возвращаемого значения методов проверки ЭП используется структура:

public class VerifyResult implements Dto {

private List<SignerInfo> signerInfos = new ArrayList<SignerInfo>();

public List<SignerInfo> getSignerInfos() {

return signerInfos;

}

public void setSignerInfos(List<SignerInfo> signerInfos) {

this.signerInfos = signerInfos;

}

}

public class SignerInfo implements Dto{

private boolean valid;

private String name;

private String certificateId;

private Date certificateValidFrom;

private Date certificateValidTo;

private String error;

private Date signDate;

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getCertificateId() {

return certificateId;

}

public void setCertificateId(String certificateId) {

this.certificateId = certificateId;

}

public Date getCertificateValidFrom() {

return certificateValidFrom;

}

public void setCertificateValidFrom(Date certificateValidFrom) {

this.certificateValidFrom = certificateValidFrom;

}

public Date getCertificateValidTo() {

return certificateValidTo;

}

public void setCertificateValidTo(Date certificateValidTo) {

this.certificateValidTo = certificateValidTo;

}

public String getError() {

return error;

}

public void setError(String error) {

this.error = error;

}

public boolean isValid() {

return valid;

}

public void setValid(boolean valid) {

this.valid = valid;

}

public Date getSignDate() {

return signDate;

}

public void setSignDate(Date signDate) {

this.signDate = signDate;

}

}

Структура VerifyResult содержит информацию о результате проверки ЭП. Подпись может содержать информацию сразу о нескольких подписывающих, для этого существует список signerInfos. Каждый SignerInfo содержит информацию о валидности подписи и информацию о подписавшем. В случае невалидности подписи будет заполнен атрибут error, который будет содержать причину невалидности подписи.

EJB вызывает спринг бин с конкретной реализацией, использующей конкретный крипто провайдер. Интерфейс спринг бина:

\*\*

\* Спринг интерфейс для выполнения криптографических операций. Может быть несколько реализация с помошью разных провайдеров

\* Конкретный бин подключается к приложению в global конфигурации

\*

\*/

public interface CryptoBean{

/\*\*

\* Проверка усовершенствованной ЭП включеннной в документ

\* @param inputStream поток содержащий документ со встроенной электронной подписью

\* @return

\*/

VerifyResult verify(InputStream document);

/\*\*

\* Проверка Усовершенствованной ЭП не включенной в документ

\* @param document поток содержащий документ

\* @param signature электронная подпись

\* @return

\*/

VerifyResult verify(InputStream document, byte[] signature);

/\*\*

\* Проверка стандартной ЭП

\* @param document поток содержащий документ

\* @param signature электронная подпись

\* @param signerSertificate сертификат подписавшено документ в формате DER

\* @return

\*/

VerifyResult verify(InputStream document, byte[] signature, byte[] signerSertificate);

/\*\*

\* Формирование hash переданного документа

\* @param document

\* @return

\*/

byte[] hash(InputStream document);

}

Возможно несколько спринг бинов использующих разные криптопровайдеры. Подключение того или иного производится в глобальной конфигурации в теге configuration.global-settings.crypto-settings в атрибуте server-component-name. Пример глобальной конфигурации crypto-settings:

<global-settings>

…

<crypto-settings gui-component-name="cryptopro.browser.plugin.client.component"

server-component-name="bouncycastleCryptoBean"

hash-on-server="true">

<provider-settings>

<cades-crypto-settings-config ts-address="http://cryptopro.ru/tsp/" />

</provider-settings>

</crypto-settings>

…

</global-settings>

Так же в crypto-settings задаются следующие параметры:

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| gui-component-name | Атрибут | Имя клиентской компоненты, которая вызывается по действию подписать. Компонента должна наследоваться от класса DigitalSignatureClientComponent. В ядре реализована одна компонента использующая для своей работы CryptoPro Browser Plugin и называется компонента cryptopro.browser.plugin.client.component, при необходимости можно разработать другой компонент и подключить его. |
| server-component-name | Атрибут | Имя серверного спринг бина, реализующий крипто операции на сервере. В платформе реализованы крипто бины наи базе CryptoPro JCP и на базе библиотеки Bouncycastle и называются cryptoProCryptoBean и bouncycastleCryptoBean соответственно. |
| hash-on-server | Атрибут | Флаг места хеширования документа. В случае true хеширование будет производится на сервере, в случае false на клиенте. Хеширование на сервере наиболее предпочтительно, так как в этом случае на клиент не пересылается весь подписываемый документ, особенно полезна эта функция в случае больших вложений. |
| provider-settings | Элемент | Произвольная настройка криптопровайдера. Текущая реализация в платформе требует настройки cades-crypto-settings-config в которой устанавливается сервер штампа времени. |

Для встраивания ЭП в карточку документа в платформе создан action с именем aDigitalSignature. Пример добавления действия ЭП в карточку:

<act:tool-bar>

…

<act:action-ref name-ref=*"aDigitalSignature"*/>

</act:tool-bar>

Функционал формирования контента для подписи и хранения контента вынесены в отдельные бины, для того чтобы предоставить разработчикам прикладных систем максимальную свободу в формирование контента и средствах хранения подписи. Данная функциональность вынесена в спринг бины и имена этих спринг бинов прописываются в конфигурации type-crypto-settings. Бин формирования контента для ЭП имеет интерфейс:

/\*\*

\* Сервис подготовки вложений к ЭП

\* @author larin

\*

\*/

public interface SignatureDataService {

/\*\*

\* Получение идентификаторов доменных объектов которые надо подписать в пачке, например все вложения к карточке документа

\* @param settings

\* @param rootId

\* @return

\*/

List<Id> getBatchForSignature(CollectorSettings settings, Id rootId);

/\*\*

\* Получение контента для подписи. Это может быть вложение, может xml представление карточки или что то иное

\* @param settings

\* @param id

\* @return

\*/

SignedDataItem getContentForSignature(CollectorSettings settings, Id id);

}

Алгоритм вызова бина следующий. При нажатие кнопки действия “Подписать” в конфигурации ищется type-crypto-settings для типа ДО, в контексте которого была нажата кнопка Action. При этом учитывается иерархия типов. У найденной конфигурации зачитывается значение атрибута get-content-bean-name и получается спринг бин с этим именем. Далее у найденного бина вызывается метод getBatchForSignature. Метод возвращает идентификаторы всех связанных ДО чьи контенты необходимо подписать. Далее по каждому полученному идентификатору опять производится поиск подходящего type-crypto-settings. По полученной конфигурации так же получается бин формирования контента и у этого бина вызывается метод getContentForSignature. В возвращаемой структуре SignedDataItem есть ссылка на поток, из которого загружается контент непосредственно для подписи. Далее из возвращенного потока формируется контент для подписи и передается на клиент для непосредственного формирования ЭП.

В платформе реализовано два бина подготовки контента для подписи:

1. allAttachmentSignatureDataService – Бин получающий все вложения связанные с типом ДО отображаемом на форме. Для получения всех вложений используется метод AttachmentService.findAttachmentDomainObjectsFor(). Для настройки бина можно использовать фильтры по типу и по атрибуту name, например чтобы из всех вложений исключить вложения определенного типа или с определенным расширением. Пример конфигурации исключающий типы type1 и type2 и вложения в имени которых есть \*.sig. Фильтр по имени поддерживает регулярные выражения:

<type-crypto-settings name="country\_attachment"

get-content-bean-name="currentAttachmentSignatureDataService"

signature-storage-bean-name="domainObjectStoreSignatureService">

<get-content-bean-settings>

<all-attachment-signature>

<exlude-attachment-type>type1</exlude-attachment-type>

<exlude-attachment-type>type2</exlude-attachment-type>

<exlude-attachment-name>\.sig</exlude-attachment-name>

</all-attachment-signature>

</get-content-bean-settings>

<signature-storage-bean-settings>

<domain-object-store-signature

signature-store-type-name="digital\_signature"

signed-attachment-field-name="signed\_object"

signature-field-name="signature"/>

</signature-storage-bean-settings>

</type-crypto-settings>

1. currentAttachmentSignatureDataService – бин получающий контент объекта с переданным идентификатором. Данный бин может работать только с вложениями. Пример конфигурации:

<type-crypto-settings name="fauna\_attachment"

get-content-bean-name="currentAttachmentSignatureDataService"

signature-storage-bean-name="domainObjectStoreSignatureService">

<signature-storage-bean-settings>

<domain-object-store-signature

signature-store-type-name="digital\_signature"

signed-attachment-field-name="signed\_object"

signature-field-name="signature"/>

</signature-storage-bean-settings>

</type-crypto-settings>

После формирования подписи на клиенте производится сохранение подписи. Клиент формирует структуру содержащую подпись в формате base64 и идентификатор подписанного объекта. По идентификатору ищется подходящий type-crypto-settings, из атрибута signature-storage-bean-name получается имя бина выполняющего сохранение подписи и у данного бина вызывается метод saveSignature()

Для проверки подписи создан акшен aVerifyDigitalSignature. Его можно добавить на карточку ДО.

<act:action-ref name-ref="aVerifyDigitalSignature"/>

При активации данного действия вызывается метод CryptoService.verify(Id). Логика работы метода CryptoService.verify(Id) следующая:

1. Получается SignatureDataService для переданного идентификатора.
2. Вызывается метод getBatchForSignature для переданного идентификатора как и при формирование подписи.
3. Для каждого возвращенного идентификатора получаются свои SignatureDataService и SignatureStorageService
4. Получается контент и подпись вызовами SignatureStorageService.getContentForSignature и SignatureStorageService.loadSignature соответственно.
5. Выполняется проверка подписи, результат проверки возвращается в структуре List<DocumentVerifyResult>.
6. В интерфейсе отобразится диалог с информациями о существующих подписях.

В платформе реализован бин для хранения подписи в атрибутах доменного объекта. Бин имеет имя domainObjectStoreSignatureService и имеет конфигурацию задающую имя типа ДО хранения подписи, имя атрибутов ссылающихсф на подписанный контент (аттачмент) и имя атрибута в котором непосредственно хранится подпись в формате base64. Пример конфигурации:

<type-crypto-settings name="fauna\_attachment"

get-content-bean-name="currentAttachmentSignatureDataService"

signature-storage-bean-name="domainObjectStoreSignatureService">

<signature-storage-bean-settings>

<domain-object-store-signature

signature-store-type-name="digital\_signature"

signed-attachment-field-name="signed\_object"

signature-field-name="signature"/>

</signature-storage-bean-settings>

</type-crypto-settings>

signature-store-type-name – имя типа где хранится подпись,

signed-attachment-field-name – имя поля , в который записывается ID доменного объекта контента

signature-field-name – Имя атрибута в который записывается подпись в виде строки в кодировке base64