Встраивание Рутокен ЭЦП 2.0/3.0, используя Рутокен Плагин

В статье описана типичная схема подобной интеграции, основанная на следующих сценариях использования Плагина:

Данные сценарии предполагают клиент-серверное взаимодействие, написание клиентских скриптов на JavaScript и соответствующих им серверных вызовов OpenSSL.

Google Chrome/Chromium

Если вы открываете локальную страницу, убедитесь, что в настройках расширения Адаптер Рутокен Плагин включена опция **"Разрешить открывать локальные файлы по ссылкам"**

Общие операции

Так как все функции Плагина выполняются в отдельных потоках, не заставляя браузер ждать выполнения операций, его интерфейс постороен на промисах. Промисы - это способ организации асинхронного кода. Универсальный метод добавления обработчиков:

Добавление Promise обработчика

```
promise.then(onFulfilled, onRejected)
// onFulfilled – ,          .
// onRejected – ,          .
```

Операции с устройством

Поиск подключенных устройств

Любой клиентский сценарий начинается с поиска подключенных к компьютеру USB-устройств Рутокен.

```
function handleError(reason) {
    if (isNaN(reason.message)) {
        alert(reason);
    } else {
        var errorCodes = plugin.errorCodes;
        switch (parseInt(reason.message)) {
            case errorCodes.PIN_INCORRECT:
                alert(" PIN");
                break;
            default:
                alert("  #" + reason.message);
        }
    }
}

// ...

plugin.enumerateDevices().then(function(devices) {
        if (devices.length > 0) {
                // ...
        } else {
                return Promise.reject("  ");
        }
}, handleError)
```

При этом вызове возвращается список идентификаторов подключенных устройств. Идентификатор представляет собой число, связанное с номером слота, к которому подключено устройство. При повторном перечислении это число может отличаться для одного и того же устройства.

Стоит заметить, что вызов этот произойдет асинхронно. Первый аргумент метода then, вызывается в случае успешного выполнения Primve enumerateDevices, второй --в случае ошибки. В данном примере был написан обобщенный обработчик ошибок. Его можно расширить и использовать в дальнейшем в своих проектах.

Рутокен Плагин определяет все подключенные к компьютеру USB-устройства Рутокен ЭЦП. Поэтому следующим шагом следует определить тип устройства.

Получение информации об устройстве

Для определения типа устройства следует использовать функцию getRutokenModelName() репозитория https://github.com/AktivCo/rutoken-model-by-hw-features-js

> *Параметр функции getDeviceInfo TOKEN_INFO_DEVICE_TYPE – устарел и не поддерживается.*

С помощью функции getDeviceInfo можно получить:

- модель устройства
- метку устройства
- серийный номер устройства
- информацию о том, залогинен ли пользователь на устройство

Смена PIN-кода

Пример смены PIN-кода на устройстве:

```
//   Promise,    Id
.then(function(rutokenHandle){
        options={}
        return plugin.changePin(rutokenHandle, "12345678", "12345678", options)
})
.then(function() {
        alert("PIN  ");
}, handleError)
```

Здесь первым параметром выступает старый PIN-код, а вторым новый PIN-код.

Работа с сертификатами

1. На токене могут храниться 3 категории сертификатов:

- пользовательские (константа в плагине CERT_CATEGORY_USER)
  Это сертификаты, связанные с закрытым ключом пользователя. Применяются, например, для подписи CMS/PKCS#7, аутентификации пользователя в протоколе TLS.
  Если сертификат импортируется как пользовательский, то при импорте будет произведен поиск в устройстве соответствующего ему закрытого ключа. Если такой ключ будет найден, то сертификат будет «привязан» к этому ключу. Если ключ найден не будет, то вернется ошибка.
- корневые (константа в плагине CERT_CATEGORY_CA)
  Это сертификаты издателей сертификатов, применяющиеся для проверки подписи под сертификатами. Подобная проверка подписи (построение цепочки доверия) позволяет определить, доверяет ли пользователь подписи другого пользователя. Например, в функции плагина verify есть режим проверки сертификата, на котором было подписано сообщение. При этом используется хранилище корневых сертификатов на токене, созданное импортом корневых сертификатов на токен.
- другие (константа в плагине CERT_CATEGORY_OTHER)
  Это сертификаты, которые не связаны с закрытым ключом и не являются корневыми.

2. Для чтения сертификатов, хранящихся на устройстве, не требуется авторизация на устройство.

Пример чтения пользовательских сертификатов с устройства:

```
//   Promise,    Id
.then(function(rutokenHandle) {
        return plugin.enumerateCertificates(rutokenHandle, plugin.CERT_CATEGORY_USER);
})
//
.then(function(certs) {
        if (certs.length > 0) {
                // ...
        } else {
                return Promise.reject("    ");
        }
}, handleError)
```

3. Сертификат можно экспортировать в PEM-формате:

```
//   Promise,    Id   Id
.then(function(rutokenHandle, certHandle) {
        return plugin.getCertificate(rutokenHandle, certHandle);
})
//      PEM
.then(function(pem) {
        alert(pem);
}, handleError)
```

Получится примерно такая строка:

```
-----BEGIN CERTIFICATE-----
MIIBmjCCAUegAwIBAgIBATAKBgYqhQMCAgMFADBUMQswCQYDVQQGEwJSVTEPMA0G
A1UEBxMGTW9zY293MSIwIAYDVQQKFBlPT08gIkdhcmFudC1QYXJrLVRlbGVjb20i
MRAwDgYDVQQDEwdUZXN0IENBMB4XDTE0MTIyMjE2NTEyNVoXDTE1MTIyMjE2NTEy
NVowEDEOMAwGA1UEAxMFZmZmZmYwYzAcBgYqhQMCAhMwEgYHKoUDAgIjAQYHKoUD
AgIeAQNDAARADKA/O1Zw50PzMpcNkWnW39mAJcTehAhkQ2Vg7bHkIwIdf7zPe2Px
HyAr6lH+stqdACK6sFYmkZ58cBjzL0WBwaNEMEIwJQYDVR0lBB4wHAYIKwYBBQUH
AwIGCCsGAQUFBwMEBgYpAQEBAQIwCwYDVR0PBAQDAgKkMAwGA1UdEwEB/wQCMAAw
CgYGKoUDAgIDBQADQQD5TY55KbwADGKJRK+bwCGZw24sdIyayIX5dn9hrKkNrZsW
detWY3KJFylSulykS/dfJ871IT+8dXPU5A7WqG4+
-----END CERTIFICATE-----
```

4. Сертификат можно распарсить вызовом функции parseCertificate и получить из него DN Subject, DN Issuer, расширения, значение открытого ключа, подпись, серийный номер, срок действия и т.п.

5. Сертификат можно записать на устройство.

Пример записи сертификата на устройство как пользовательского:

```
 var certpem = "-----BEGIN CERTIFICATE-----
MIIBmjCCAUegAwIBAgIBATAKBgYqhQMCAgMFADBUMQswCQYDVQQGEwJSVTEPMA0G
A1UEBxMGTW9zY293MSIwIAYDVQQKFBlPT08gIkdhcmFudC1QYXJrLVRlbGVjb20i
MRAwDgYDVQQDEwdUZXN0IENBMB4XDTE0MTIyMjE2NTEyNVoXDTE1MTIyMjE2NTEy
NVowEDEOMAwGA1UEAxMFZmZmZmYwYzAcBgYqhQMCAhMwEgYHKoUDAgIjAQYHKoUD
AgIeAQNDAARADKA/O1Zw50PzMpcNkWnW39mAJcTehAhkQ2Vg7bHkIwIdf7zPe2Px
HyAr6lH+stqdACK6sFYmkZ58cBjzL0WBwaNEMEIwJQYDVR0lBB4wHAYIKwYBBQUH
AwIGCCsGAQUFBwMEBgYpAQEBAQIwCwYDVR0PBAQDAgKkMAwGA1UdEwEB/wQCMAAw
CgYGKoUDAgIDBQADQQD5TY55KbwADGKJRK+bwCGZw24sdIyayIX5dn9hrKkNrZsW
detWY3KJFylSulykS/dfJ871IT+8dXPU5A7WqG4+
-----END CERTIFICATE-----";

// ...

//   Promise,    Id   Id
.then(function(rutokenHandle, certPem) {
        return plugin.importCertificate(rutokenHandle, certPem, plugin.CERT_CATEGORY_USER);
})
//      PEM
.then(function() {
        alert("  ");
}, handleError)
```

6. Вызовом функции deleteCertificate можно удалить сертификат с токена.

Работа с ключевыми парами ГОСТ

1. Для получения дескрипторов ключевых пар, хранящихся на устройстве, требуется ввод PIN-кода. Следует понимать, что само значение закрытого ключ получено быть не может, так как ключ является неизвлекаемым.

```
//   Promise,    Id
.then(function(rutokenHandle) {
        return plugin.enumerateKeys(rutokenHandle, "");
})
.then(function(keys) {
        if (keys.length > 0) {
                // ...
        } else {
                return Promise.reject("    ");
        }
}, handleError)
```

2. Для генерации ключевой пары требуется ввод PIN-кода. При генерации ключа можно задать параметры ключевой пары. Если опция не задана, будут выбраны параметры A для любого из алгоритмов ГОСТ.

- ГОСТ Р 34.10-2012 длина закрытого ключа 256 бит:
  - "A"
  - "B"
  - "C"
  - "XA"
  - "XB"
- ГОСТ Р 34.10-2012 длина закрытого ключа 512 бит
  - "A"
  - "B"

Пример генерации ключевой пары ГОСТ Р 34.10-2012:

```
//   Promise,    Id
.then(function(rutokenHandle) {
        return plugin.generateKeyPair(rutokenHandle, undefined, marker, options);
})
.then(function(keyId) {
        alert("   ");
}, handleError)
```

3. С помощью функции deleteKeyPair ключевая пара может быть удалена с токена.

Создание и проверка электронной подписи

1. Для создание подписи используется метод sign:

Подпись данных

```
//
.then(function(certHandle, textToSign) {
var options = {};
return plugin.sign(rutokenHandle, certHandle, textToSign, plugin.DATA_FORMAT_PLAIN, options);
)
```

Последние аргумент – опции функции. Их подробный список можно найти в документации, некоторые из них:

- *detached:bool (false)* - генерировать отсоединенную подпись
- *addUserCertificate:bool (true)* - включить в подпись сертификат пользователя
- *addSignTime:bool (false)* - включить в подпись время подписи
- *useHardwareHash:bool (false)* - производить аппаратное хеширование данных на ключах ГОСТ
- *rsaHashAlgorithm:enum* - алгоритм хеширования при использовании ключей RSA, варианты: HASH_TYPE_MD5, HASH_TYPE_SHA1, HASH_TYPE_SHA256, HASH_TYPE_SHA512. *Нужно обязательно указывать при подписи на RSA ключах.*

Предпоследний аргумент определяет формат подписываемых данных:

- *DATA_FORMAT_PLAIN*. Используется для подписи незакодированных данных.
- *DATA_FORMAT_BASE64* Используется для подписи данных в base64 формате.
- *DATA_FORMAT_HASH*. Используется для подписи готового хеша. *Для корректной работы необходимо выставить опцию detached в true.*

2. Для проверки подписи используется метод verify:

Подпись данных

```
//
.then(function(certPem, CA, cms) {
var options = {certificates: [certPem], CA: CA};
return plugin.verify(rutokenHandle, cms, options);
})
```

Последние аргумент – опции функции. Их подробный список можно найти в документации, некоторые из них:

- *data:string (null)* - подписанные данные (текстовые или base64-encoded), только в случае detached подписи;
- *base64:bool (false)* - указывает, закодированы ли данные, переданные в data, в base64;
- *certificates:string[] (null)* - набор сертификатов в PEM формате, на которых необходимо проверять подпись, при этом сертификаты, которые содержатся в cms, будут проигнорированы;
- *CA:string[] (null)* - список дополнительных корневых сертификатов в PEM формате для проверки сертификата, кроме них берутся сертификаты с устройства;
- *CRL:string[] (null)* - список отозванных сертификатов в PEM формате;
- *useHardwareHash:bool (false)* - производить хеширование на устройстве (игнорируется для алгоритмов отличных от ГОСТ Р 34.10-2001);
- v*erifyCertificate:bool (true)* - проверить сертификат пользователя;

Конфигурирование openssl

См. инструкцию Интеграция ГОСТ 2012 с Рутокен ЭЦП и OpenSSL 1.1.0 или новее, раздел Установка и настройка OpenSSL для работы с rtengine 0.7.x

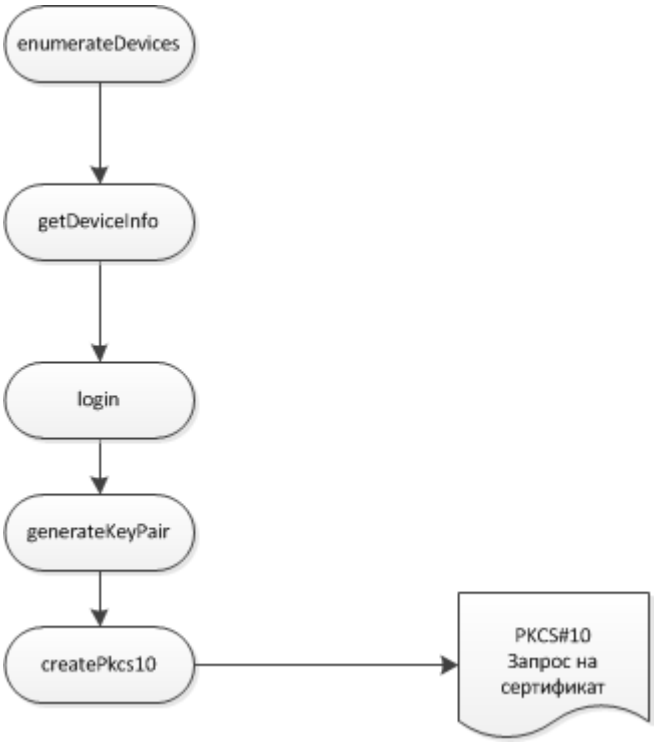
Теперь перейдем к реализации законченных пользовательских сценариев.

Регистрация на портале

Сертификат выдается при регистрации в системе

- Получаем список подключенных к компьютеру устройств Рутокен ЭЦП 2.0
- Генерируем ключевую пару по ГОСТ Р 34.10-2012 на выбранном Рутокен ЭЦП 2.0
- Создаем запрос PKCS#10 на сертификат для сгенерированной ключевой пары
- Отправляем запрос на сервер
- На сервере создаем сертификат, привязываем к аккаунту (сам сертификат или его дескриптор). Следует отметить, что дескрипторы сертификатов, полученные при вызове функции enumerateCertificates, являются уникальными и неизменными
- Отправляем сертификат на клиент
- На клиенте визуализируем полученный сертификат
- Импортируем полученный сертификат в Рутокен ЭЦП 2.0

Последовательность вызовов в клиентском скрипте будет следующей:



Далее запрос отправляется на сервер, где на его основе выдается сертификат.
Для этого на сервере должен быть установлен и правильно сконфигурирован openssl версии от 1.0 и развернут функционал УЦ.

1. Генерация улюча УЦ:

```
openssl genpkey -algorithm gost2012_256 -pkeyopt paramset:A -out ca.key
```

После этого в файле ca.key будет создан закрытый ключ

2. Создание самоподписанного сертификата УЦ:

```
openssl req -utf8 -x509 -key ca.key -out ca.crt
```

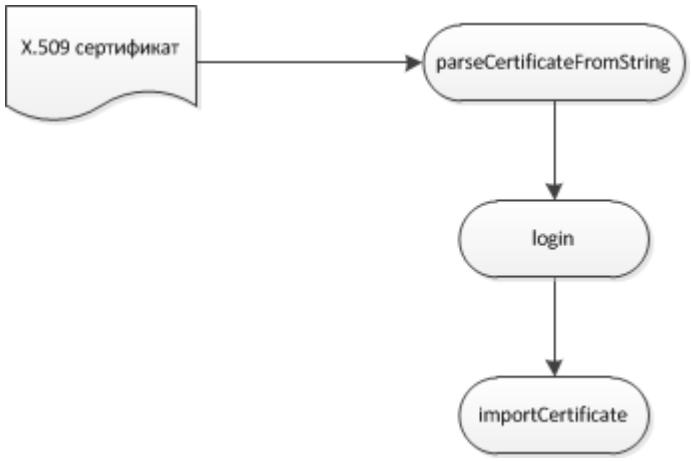
После ввода необходимой информации об издателе в файле ca.crt будет создан сертификат УЦ.

Полученный от клиента запрос сохраняем в файл user.csr и выдаем на его основе сертификат (без модификации данных из запроса):

```
openssl ca -keyfile ca.key -cert ca.crt -in user.csr -out user.crt -batch
```

После этого в файле user.crt создается сертификат пользователя в PEM формате. Его следует отправить на клиент.

Дальнейшая последовательность вызовов на клиенте:

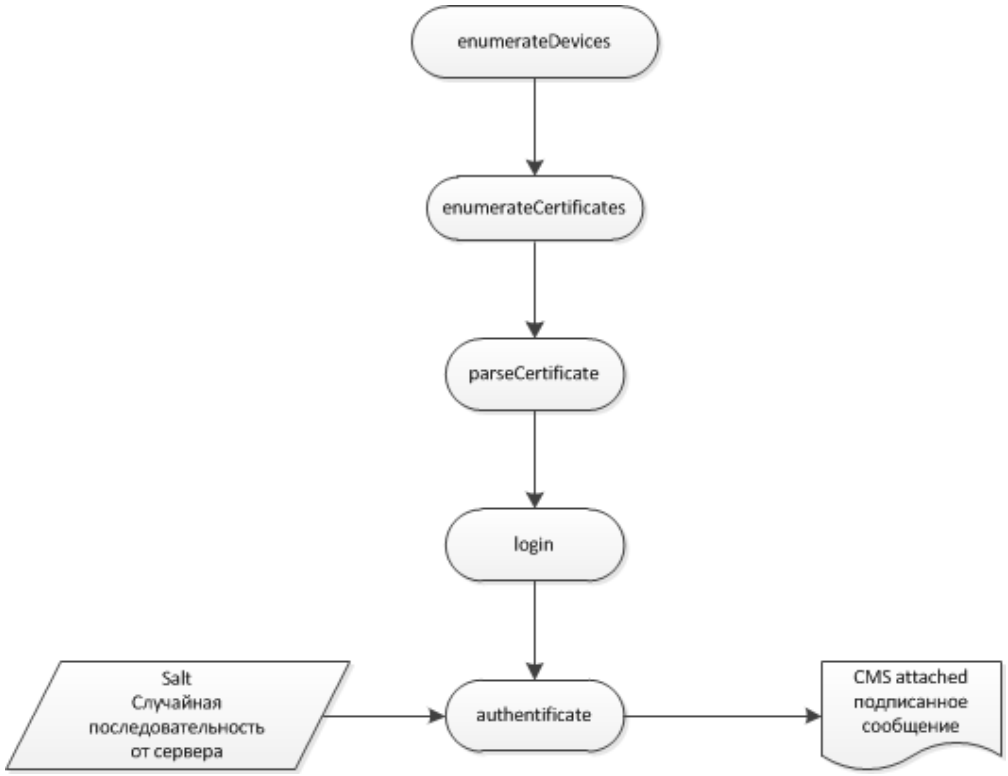


Сертификат уже имеется на токене, выдан внешним УЦ

Ключевая пара при этом должна быть создана в формате, совместимом с библиотекой rtPKCS11ECP для Рутокен ЭЦП 2.0.

- Получаем список подключенных к компьютеру устройств Рутокен ЭЦП 2.0
- Получаем список всех имеющихся пользовательских сертификатов на выбранном Рутокен ЭЦП 2.0
- Визуализируем каждый сертификат
- Пользователь выбирает нужный сертификат
- Сервер формирует начальную последовательность случайных данных (строку salt) и отправляет ее на клиент
- Вызываем на клиенте authenticate. При передаче salt в функцию плагина authenticate данная последовательность дополняется дополнительными случайными данными размером в 32 символа, и происходит подпись итоговой последовательности на выбранном пользователем сертификате в формате CMS attached
- Подпись отправляется на сервер
- На сервере происходит проверка CMS attached подписи с использованием корневого сертификата
- Из CMS attached сообщения извлекается итоговая случайная последовательность, “отсоединяется” salt и происходит сравнение
- Если сравнение успешно, то регистрируем пользователя по сертификату, который содержится в CMS attached сообщении

Последовательность вызовов на клиенте:



Подпись получается в base64-формате. При проверке ее на сервере с помощью openssl подпись следует обрамить заголовками, чтобы сделать из нее PEM. Выглядеть подобная подпись будет примерно так:

```
-----BEGIN CMS-----
MIIDUQYJKoZIhvcNAQcCoIIDQjCCAz4CAQExDDAKBgYqhQMCAgkFADCBygYJKoZI
hvcNAQcBoIG8BIG5PCFQSU5QQURGSUxFIFVURjg+PFY+0JLRi9C/0L7Qu9C90LjR
gtGMINCw0YPRgtC10L3RgtC40YTQuNC60LDRhtC40Y4/PCE+c2VydmVyLXJhbmRv
bS1kYXRhZTI6ZGE6MmM6MDU6MGI6MzY6MjU6MzQ6YzM6NDk6Nzk6Mzk6YmI6MmY6
YzU6Mzc6ZGI6MzA6MTQ6NDQ6ODM6NjY6Njk6NmI6OWY6YTU6MDk6MzQ6YmY6YzQ6
NzY6YzmgggGeMIIBmjCCAUegAwIBAgIBATAKBgYqhQMCAgMFADBUMQswCQYDVQQG
EwJSVTEPMA0GA1UEBxMGTW9zY293MSIwIAYDVQQKFBlPT08gIkdhcmFudC1QYXJr
LVRlbGVjb20iMRAwDgYDVQQDEwdUZXN0IENBMB4XDTE0MTIyMjE2NTEyNVoXDTE1
MTIyMjE2NTEyNVowEDEOMAwGA1UEAxMFZmZmZmYwYzAcBgYqhQMCAhMwEgYHKoUD
AgIjAQYHKoUDAgIeAQNDAARADKA/O1Zw50PzMpcNkWnW39mAJcTehAhkQ2Vg7bHk
IwIdf7zPe2PxHyAr6lH+stqdACK6sFYmkZ58cBjzL0WBwaNEMEIwJQYDVR0lBB4w
HAYIKwYBBQUHAwIGCCsGAQUFBwMEBgYpAQEBAQIwCwYDVR0PBAQDAgkkMAwGA1Ud
EwEB/wQCMAAwCgYGKoUDAgIDBQADQQD5TY55KbwADGKJRK+bwCGZw24sdIyayIX5
dn9hrKkNrZsWdetWY3KJFylSulykS/dfJ871IT+8dXPU5A7WqG4+MYG7MIG4AgEB
MFkwVDELMAkGA1UEBhMCUlUxDzANBgNVBAcTBk1vc2NvdzEiMCAGA1UEChQZT09P
ICJHYXJhbnQtUGFyay1UZWxlY29tIjEQMA4GA1UEAxMHVGVzdCBDQQIBATAKBgYq
hQMCAgkFADAKBgYqhQMCAhMFAARAco5PumEfUYVcLMb1cnzETNOuWC8Goda8pdUL
W5ASK+tztCwM7wpXgAy+Y6/sLtClO9sh8dKnAaEY2Yavg3altQ==
-----END CMS-----
```

Проверка подписи на сервере:

```
openssl cms -verify -in sign.cms -inform PEM -CAfile ca.crt -out data.file -certsout user.cr
```

Здесь sign.cms — файл, в котором находится подпись, ca.crt — файл с корневыми сертификатами, до одного из которых должна выстроиться цепочка, data.file — файл, в который будет сохранены подписанные данные, user.crt — файл, в который будет сохранен пользовательский сертификат. Именно из data.file нужно извлечь данные отсоединить последние 32 символа и сравнить salt.

Если на сервере нужно получить информацию из сертификата, то парсить его можно так:

Показать содержимое сертификата в текстовом представлении:

```
openssl x509 -in cert.pem -noout -text
```

Показать серийный номер сертификата:

```
openssl x509 -in cert.pem -noout -serial
```

Показать DN субъекта (subject):

```
openssl x509 -in cert.pem -noout -subject
```

Показать DN издателя (issuer):

```
openssl x509 -in cert.pem -noout -issuer
```

Показать почтовый адрес субъекта:

```
openssl x509 -in cert.pem -noout -email
```

Показать время начала действия сертификата:

```
openssl x509 -in cert.pem -noout -startdate
```

Показать время окончания действия сертификата:

```
openssl x509 -in cert.pem -noout -enddate
```

Строгая аутентификация на портале

Общая схема аутентификации, используемая в Рутокен Плагин, выглядит следующим образом:

- сервер формирует начальную последовательность случайных данных (строку salt) и отправляет ее на клиент
- при передаче salt в функцию плагина authenticate данная последовательность дополняется случайными данными размером в 32 символа, и происходит подпись итоговой последовательности на выбранном пользователем сертификате в формате CMS attached
- подпись отправляется на сервер
- на сервере происходит проверка подписи
- из CMS attached сообщения извлекается итоговая случайная последовательность, “отсоединяется” salt и происходит сравнение

- в случае успешной проверки пользователь аутентифицируется на основе сертификата, извлеченного из сообщения CMS

Реализация данной схемы ничем принципиально не отличается от «Регистрация, сертификат уже имеется, выдан внешним УЦ».

Электронная подпись данных и/или файлов в формате CMS

- формируется текстовое сообщение (строка), формирование сообщения может происходить как на сервере, так и на клиенте
- если требуется подписать документ произвольного формата (например, PDF), то требуется перекодировать его в формат base64
- строка, содержащая данные для подписи, передается в функцию sign
- если строка представляет собой закодированные в base64 данные, то параметр функции isBase64 должен быть установлен в true, при этом перед подписью произойдет декодирование данных из base64
- если требуется использовать аппаратное вычисление хэш-функции ГОСТ Р 34.11-94 (сертифицированная реализация, скорость 60-70 Кб/с), то в options нужно установить опцию useHardwareHash в true. Если данная опция установлена в false, то будет использована быстрая программная реализация хэш-функции ГОСТ Р 34.11-2012
- если требуется сформировать “отсоединенную” (detached) подпись CMS, то нужно установить опцию detached в true, иначе будет сформирована “присоединенная” (attached) подпись
- для того, чтобы включить/не включить пользовательский сертификат в подписанное CMS-сообщение существует опция addUserCertificate
- Установка опции addSignTime в true приведет к тому, что в подписанное CMS-сообщение будет добавлено системное время в качестве подписанного атрибута

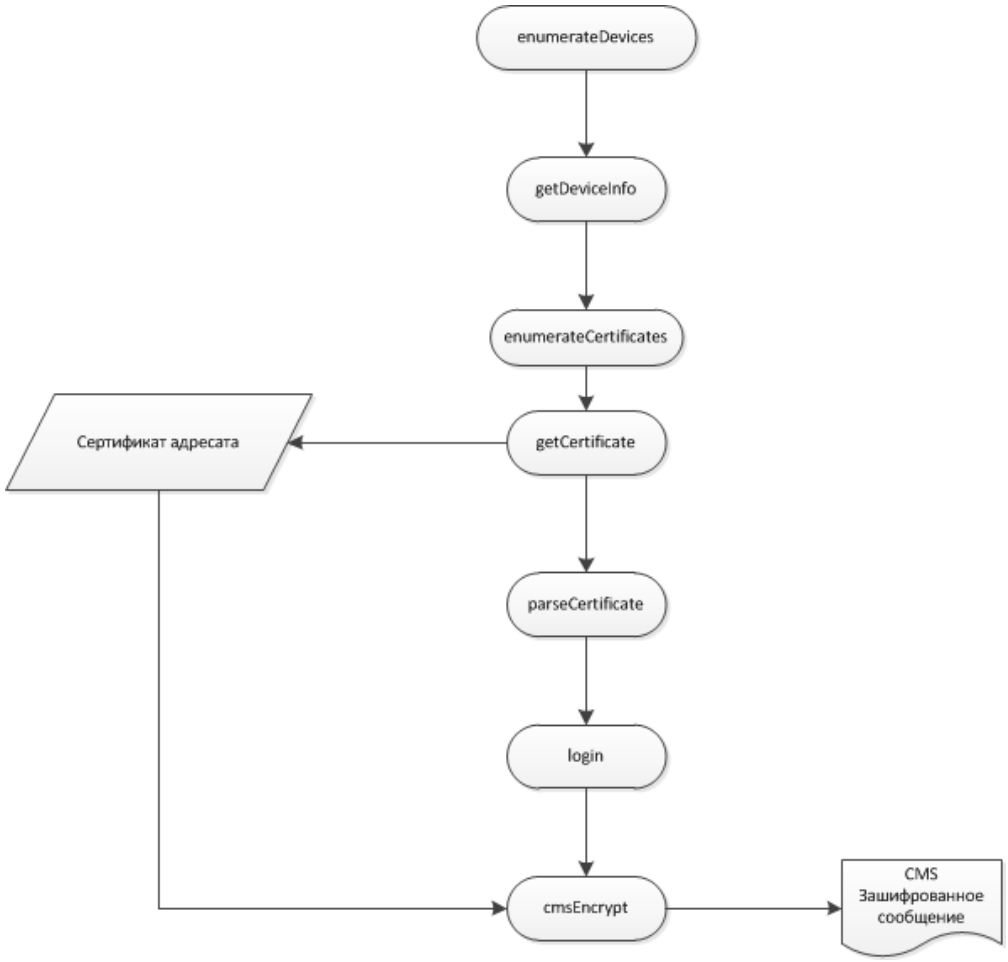
Проверка подписи на сервере описана выше.

Шифрование/расшифрование данных и/или файлов в формате CMS

Шифрование данных на клиенте для сервера

Для того, чтобы обеспечить конфиденциальность обмена данными между клиентом и сервером в плагине предусмотрено шифрование /расшифрование данных. Данные шифруются в формате CMS. Для того, чтобы зашифровать данные в формате CMS, требуется сертификат открытого ключа «адресата». При этом расшифровать такое сообщение сможет только владелец закрытого ключа. При шифровании данных для сервера рекомендуется хранить сертификат сервера на Рутокен ЭЦП 2.0. Этот сертификат может быть записан на устройство при регистрации пользователя на портале. Для этого следует использовать функцию importCertificate, при этом в качестве параметра category следует передать CERT_CATEGORY_OTHER. Для использования в функции cmsEncrypt нужно получить тело сертификата по его дескриптору с помощью функции getCertificate. При этом дескриптор является уникальным и неизменным и может быть сохранен в учетной записи пользователя на сервере при импорте сертификата сервера. Для того, чтобы использовалось аппаратное шифрование по ГОСТ 28147-89, требуется установить опцию useHardwareEncryption в true. В противном случае будет использована быстрая программная реализация ГОСТ 28147-89.

Последовательность вызовов приведена на картинке:



Шифрование данных на клиенте:

```
var data = "some data"

var recipientsCertsInPem = [
String.raw`-----BEGIN CERTIFICATE-----
MIICEjCCAb+gAwIBAgIJAL+E0An4CJgVMAoGCCqFAwcBAQMCMHkxCzAJBgNVBAYT
AlJVMQ8wDQYDVQQIDAZSdXNzaWExDzANBgNVBAcMBk1vc2NvdzEXMBUGA1UECgwO
WkFPIEFrdGl2LVNvZnQxEDAOBgNVBAsMB1J1dG9rZW4xHTAbBgNVBAMMFFJ1dG9r
ZW4gVEVTVCBDQSBHT1NUMB4XDTIwMTAxOTE3MjIyN1oXDTIxMTAxOTE3MjIyN1ow
MjEMMAoGA1UEAwwDbWVtMQswCQYDVQQGEwJSVTEVMBMGA1UECAwM0JzQvtGB0LrQ
stCwMGYwHwYIKoUDBwEBAQEwEwYHKoUDAgIjAQYIKoUDBwEBAgIDQwAEQIyKtSP4
WYjewCNr52fNMYDVwFPIwWgWdOi+DpJ47pBn4g6rILdZfvkAfemMs7a74is+mPyg
401mjbZPhDXCRt2jajBoMAsGA1UdDwQEAwIGwDATBgNVHSUEDDAKBggrBgEFBQcD
BDATBgNVHSAEDDAKMAgGBiqFA2RxATAvBgUqhQNkbwQmDCTQodCa0JfQmCAi0KDR
g9GC0L7QutC10L0g0K3QptCfIDIuMCIwCgYIKoUDBwEBAwIDQQCJvbdPKR9QO4zP
Xs4SK/dhzNdNzYmyCOoi7qUBYc4laKwUnhV88ENR5+VDee05w+JMIIJFuelg/QN3
W/W6SC2v
-----END CERTIFICATE-----`
]

//   Promise,    Id
.then(function (rutokenHandle) {
    var options = {};
        return plugin.cmsEncrypt(rutokenHandle, "", recipientsCertsInPem, data, options);
})
.then(function(msg) {
        alert(msg);
}, handleError)
```

Перед расшифрованием сообщение нужно обрамить PEM-заголовками "-----BEGIN PKCS7-----" и "-----END PKCS7-----":

Расшифрование данных на сервере

```
openssl cms -decrypt -binary -in message.cms -inform PEM -recip respondent.cer -inkey recipient.key -out
drecipient.crt
```

recipient.crt — сертификат того, для кого зашифровано сообщение, recipient.key — ключ того, для кого зашифровано сообщение.

Расшифрование данных, полученных с сервера, на клиенте

Для расшифрования данных, полученных с сервера, предназначена функция cmsDecrypt. Так как сервер шифрует для клиента, используя его сертификат, то в качестве keyId должен быть передан дескриптор закрытого ключа клиента, соответствующий открытому ключу в сертификате. Этот дескриптор является уникальным и неизменным и потому может быть сохранен в учетной записи пользователя на сервере. Кроме того, дескриптор ключа пользователя может быть получен явным образом, путем вызова функции getKeyByCertificate.

Шифрование данных на сервере для клиента:

```
openssl cms -encrypt -binary -gost28147-paramset_a-cfb -in data.file -out message.enc -outform PEM user.crt
```

Расшифрование данных на клиенте:

```
//   Promise.  rutokenHandle  keyHandle     .
.then(function () {
    var options = {};
    return plugin.cmsDecrypt(rutokenHandle, keyHandle, cms, options);
})
.then(function(msg) {
    alert(msg);
}, handleError)
```

Работа со штампами времени

Последовательность действий для добавления метки времени следующая:

1. Вызов plugin.**sign**() для получения CMS-подписи.
2. Вызов plugin.**createTsRequest**(). Если на вход передается CMS, data == DATA_FORMAT_BASE64
3. Отправка запроса TSA по HTTP
   С TSA возможно взаимодействовать по HTTP, согласно RFC: Time-Stamp Protocol via HTTP. Это обычный POST-запрос.
   Возможно, на TSA-сервере потребуется разрешить запросы с домена, откуда клиент будет посылать запрос, используя JS. Также нужно будет использовать CORS-заголовки.
4. Получение ответа от TSA
5. Вызов функции для проверки ответа TSA plugin.**verifyTsResponse()**
6. Для добавления метки времени в CMS, мы предоставляем функцию в исходных кодах, использующую библиотеку PKIjs:
   cmsWithTimeStamp = **asn1Utils.addTstToSignedCms(**signedCms, tsResp);

Полезные ссылки

Данные ссылки могут быть полезны разработчикам инфосистем с поддержкой ЭП на базе Рутокен Плагин и OpenSSL:

Пример интеграции Рутокен Плагин в ДБО
Веб-сервис генерации ключей, формирования запросов, управления сертификатами
Документация по использованию утилиты OpenSSL с российскими криптоалгоритмами