Руководство разработчика

Для того, чтобы начать использовать pki-core, необходимо в настройках проекта добавить путь до каталога с заголовочными файлами библиотеки и включить файл rutoken/pki-core-cpp.h. Приложение должно быть слинковано с динамической библиотекой pki-core.

Первым вызовом pki-core-cpp должен быть `rutoken::pkicore::initialize` с указанием каталога, в котором лежит библиотека rtPKCS11ECP. По завершении работы с pki-core-cpp необходимо вызвать `rutoken::pkicore::deinitialize` для освобожения ресурсов библиотеки.

Жизненный цикл объектов

Большая часть классов, представленых в библиотеке, описывает объекты предметной области, не подразумевающие возможность копирования (нельзя на устройстве создать копию сертификата или ключа подписи). Такие классы имеют move-конструкторы, которые можно использовать для перемещения объектов.

Базовая работа с устройствами и сертификатами

pki-core-cpp позволяет работать с токенами, доступными через библиотеку rtPKCS11ECP. Для перечисления подключенных устройств необходимо использовать функцию `rutoken::pkicore::Pkcs11Device::enumerate`, которая возвращает список объектов класса `rutoken::pkicore::Pkcs11Device`. Вызов этой функции делает объекты Pkcs11Device, полученные в предыдущем вызове, недействительными. Также недействительными становятся объекты ключей и сертификатов, хранящихся на устройствах.

Для перечисления и использования ключевых пар на токене требуется предварительно авторизоваться, используя функцию `rutoken::pkicore::Pkcs11Device::login`. Чтобы сбросить права доступа на токене, необходимо вызвать функцию `rutoken::pkicore::Pkcs11Device::logout`.

Большинство операций, связанных с PKI, используют объекты классов `rutoken::pkicore::ExternalCert`, `rutoken::pkicore::Pkcs11Cert`, `rutoken::pkicore::Pkcs11UserCert`. Объекты класса ExternalCert используются, когда необходимо передать какой-либо сертификат, не хранящийся на токене. Используя метод `rutoken::pkicore::Pkcs11Device::enumerateCerts`, можно получить все сертификаты, хранящиеся на устройстве. Метод `rutoken::pkicore::Pkcs11Device::enumerateUserCerts` возвращает пользовательские сертификаты на устройстве, которые могут использоваться в операциях, требующих наличия соответствующей ключевой пары.

```
using namespace rutoken::pkicore;

initialize(".");

auto devices = Pkcs11Device::enumerate();

//   .

assert(devices.size() == 1);
auto device = std::move(devices.front());

auto certs = device.enumerateCerts();

device.login("12345678");

auto userCerts = device.enumerateUserCerts();

device.logout();

deinitialize();
```

Формирование запроса PKCS#10

В общем случае для получения пользовательского сертификата, который можно использовать в операциях подписи, необходимо выполнить следующие действия:

- Сгенерировать на устройстве ключевую пару, которая будет использоваться для подписи, с помощью метода `rutoken::pkicore::Pkcs11Device::generateKeyPair`.
- Заполнить необходимые поля PKCS#10 запроса на сертификат, используя класс `rutoken::pkicore::Pkcs10RequestInfo`.
- Сформировать PKCS#10 запрос, вызвав функцию `rutoken::pkicore::createPkcs10Request`.
- Получить сертификат в центре сертификации на основании запроса, полученного на предыдущем шаге.

- Импортировать полученный сертификат на токен, используя метод `rutoken::pkicore::Pkcs11Device::importCert`.

```
using namespace rutoken::pkicore;

auto key = device.generateKeyPair(Pkcs11Device::Gost34102001KeyGenParams());

X500Dn subject;
subject.setRdn(X500Dn::RdnId::commonName, "Ivanov Ivan Ivanovich");

Pkcs10RequestInfo info;
info.setSubject(subject);

auto request = createPkcs10Request(key, info);

//  request.toPem()  PKCS#10      PEM,          .

device.importCert(ExternalCert(pemCert));
```

Подпись и проверка подписи в формате CMS

Для создания подписанного CMS сообщения необходимо использовать функцию `rutoken::pkicore::cms::sign`. На данный момент поддерживается подпись объектов с типом Data Content, который может содержать произвольные данные. Параметры операции подписи задаются через класс `rutoken::pkicore::cms::SignParams`. Используя метод `rutoken::pkicore::cms::SignedData::toBer`, можно получить сообщение в формате BER.

Статический метод `rutoken::pkicore::cms::SignedData::parse` позволяет получить объект класса SignedData из сообщения в формате BER. Подпись сообщения можно проверить, используя метод `rutoken::pkicore::cms::SignedData::verify`. Параметры проверки задаются классом `rutoken::pkicore::cms::VerifyParams`.

```
using namespace rutoken::pkicore;

std::vector<unsigned char> data(10, 'a');

//  SignParams    ,        .
auto signedData = cms::sign(cms::Data(data.begin(), data.end()), cms::SignParams(cert));

//  VerifyParams   ,      .
auto result = signedData.verify(cms::VerifyParams(device));

assert(result == cms::VerifyResult::success);
```

Шифрование и расшифрование в формате CMS

Для создания шифрованного CMS сообщения необходимо использовать функцию `rutoken::pkicore::cms::envelop`. На данный момент поддерживается шифрование объектов с типом Data Content, который может содержать произвольные данные. Параметры операции шифрования задаются через класс `rutoken::pkicore::cms::EnvelopParams`. Используя метод `rutoken::pkicore::cms::EnvelopedData::toBer`, можно получить сообщение в формате BER.

Статический метод `rutoken::pkicore::cms::EnvelopedData::parse` позволяет получить объект класса EnvelopedData из сообщения в формате BER. Расшифровать сообщение можно, используя метод `rutoken::pkicore::cms::EnvelopedData::decrypt`. Параметры расшифрования задаются классом `rutoken::pkicore::cms::EnvelopedData::DecryptParams`.

```
using namespace rutoken::pkicore;

std::vector<unsigned char> data(10, 'a');

//  EnvelopParams   ,     ,     .
auto envelopedData = cms::envelop(cms::Data(data.begin(), data.end()), cms::EnvelopParams(device, ExternalCert
(pemCert)));

//  DecryptParams    ,        .
auto message = envelopedData.decrypt(cms::EnvelopedData::DecryptParams(userCert));

//        Data Content.
assert(message.contentType() == cms::ContentType::data);

//     .
auto result = cms::Data::cast(std::move(message));
auto resultData = result.data();

assert(resultData.size() == data.size());
assert(std::equal(data.begin(), data.end(), resultData.begin()));
```