Взаимная аутентификация Партнера и Банка (mTLS)

АО «Береке Банк» для работы с Партнерами использует mTLS — взаимную аутентификацию клиента и сервера при передаче данных. Т.е. Банк удостоверяется в подлинности подключения Партнера, а Партнер удостоверяется в подлинности подключения к Банку. Протокол mTLS является расширением TLS, который обеспечивает надежное шифрование данных и исключает возможность атаки вида «Man In the Middle» (человек посередине или прослушивания трафика).

mTLS использует сертификаты SSL как на стороне Банка так и на стороне Партнера. Для исключения какой-либо возможности утечки данных Партнер формирует свой сертификат SSL на своей стороне самостоятельно. И, в дальнейшем, сам заботится о сохранности и актуальности своего сертификата SSL.

Для создания собственного сертификата Партнером и Банком проводятся следующий ряд действий:

- Партнером формируется пара ключей (закрытый и открытый ключ RSA) достаточной длины (2048 бит) с помощью программного обеспечения на стороне Партнера или посредством сети Интернет;
- Закрытый ключ никогда не покидает информационных систем Партнера и используется им в дальнейшем для формирования запросов в Банк;
- Из открытого ключа Партнер формирует так называемый «запрос на подпись» (CSR), в котором указывает сведения о себе, как Партнере, в том объеме, который достаточен для идентификации его как Партнера. А также дополнительные, необязательные сведения, которые будут фигурировать в его сертификате;
- Созданный «запрос на подпись» CSR файл Партнер отсылает в Банк любым доступным способом. Банк, в свою очередь, произведя все проверки и согласования, подписывает его своей электронной цифровой подписью и формирует сертификат Партнера. Сертификат Партнера доставляется Партнеру в зашифрованном архиве (zip файл) посредством электронной почты. Пароль для открытия архива отправляется посредством СМС на номер Партнера. Дополнительно в том же архиве Партнеру отправляется корневой сертификат Банка;
- Партнер, получив и расшифровав свой сертификат, корневой сертификат Банка и закрытый ключ самого первого шага использует их для формирования запросов в Банк. При этом Партнер может проверять (проводить аутентификацию) стороны Банка на свой стороне. Банк же проводит аутентификацию Партнера на своей стороне. При этом Партнеру становятся доступны только те информационные системы, к которым он запрашивал доступ

Следует учесть следующее:

- Существует тестовый и продуктовый контур. Каждый контур обладает своими экземплярами корневого сертификата, сертификатом и адресом веб сайта, и требует отдельного подключения Партнера;
- Взаимная аутентификация Партнера и Банка полагается на обособленный, самоподписанный корневой сертификат Банка и требует установки его в хранилище корневых сертификатов на стороне Партнера;
- Сертификат имеет строго действующие границы по времени его действия — даты «годен от» и «годен до» и эти даты не могут быть изменены;

- Сертификат Партнеру для продуктовой среды выдается на период действия договора Партнера и Банка;
- В случае утери или компрометации закрытого ключа Партнером существует процесс отзыва действующего сертификата с последующей выдачей нового сертификата;
- Сертификат выдается Партнеру и идентифицирует Партнера по его действующим идентификаторам БИН или ИИН. Следовательно, в случае их изменения - сертификат следует перевыпустить;
- В случае изменения состава информационных систем, к которым обращается Партнер, или изменения их версий существует процесс изменения доступа действующего сертификата без необходимости выпуска нового сертификата;
- Сертификат отзывается Банком по истечении срока его действия;
- Сертификат может быть отозван или приостановлен Банком в одностороннем порядке до истечения срока его действия по причинам нарушения безопасности, подозрения или утечки данных и иных причинах;
- Закрытый ключ Партнера не передается Банку и является ответственностью Партнера по его сохранению от утечек и сохранности с течением времени;
- Партнер может формировать столько сертификатов, сколько ему необходимо. То есть применять любую политику управления сертификатами — от «один сертификат на все информационные системы Банка» до «несколько сертификатов на каждую информационную систему каждому департаменту». Идентификация Партнера происходит на уровне БИН/ИИН Партнера без дальнейшей детализации и покрывается договором между Партнером и Банком. Доступ на определенные информационные системы Банка определяется в заявке на подключение и привязывается к конкретному сертификату;
- Информационные системы Банка имеют собственные технические ограничения по частоте запросов. И, так как разделяются между всеми Партнерами, то возможна ситуация превышения частоты запросов к конкретной информационной системе. В этом случае Партнеру сразу вернется HTTP ответ 503 (Система временно недоступна). Этот статус предполагает повторный запрос через некоторое время

Создание запроса на подпись (CSR)

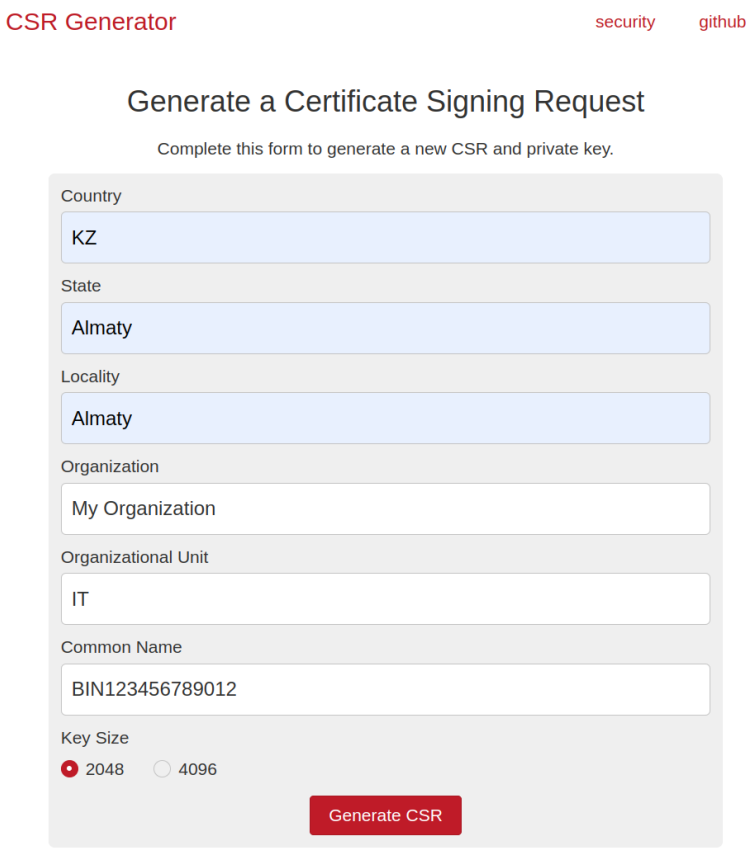
Запрос на подпись может быть создан множеством способов. От самого простого — ресурсы онлайн в сети Интернет (самый ненадежный), до генерации запроса на собственной системе (самый надежный).

Общее требование Банка к запросам на подпись только одно: в поле CN (CommonName) укажите ваш идентификатор (БИН, ИИН, ИНН, ОГРН) в соответствующем формате BIN123456789012 или IIN123456789012 и так далее, **без каких либо** лишних символов или пробелов. Иные идентификаторы приниматься **не будут**.

Страна	Тип идентификации	Формат	Пример
Казахстан	ИИН (физ. лицо) 12 цифр	IIN(12цифр)	IIN012345678901
	БИН (юр. лицо) 12 цифр	BIN(12цифр)	BIN012345678901
Россия	ИНН (физ. лицо) 12 цифр	INN(12цифр)	INN012345678901
	ИНН (юр. лицо) 10 цифр	INN(10цифр)	INN0123456789
	ОГРН (юр. лицо) 13 цифр	OGRN(13цифр)	OGRN0123456789012

Формирование CSR онлайн

В сети Интернет откройте ссылку в браузере https://csrgenerator.com/ . Заполните поля свойствами вашей организации — Партнера. При этом обратите внимание на Common Name и заполните его по шаблону, указанному выше. Размер ключа (Key Size) оставьте 2048 бит. Этого вполне достаточно, а 4096 будет заметно медленнее обрабатываться чем 2048 бит.



При нажатии на кнопку Generate CSR сайт вернет вам поле с текстом, состоящим из двух визуальных блоков. Скопируйте и сохраните этот текст в пустой текстовый файл.

Далее, скопируйте из него блок, который начинается с -----BEGIN CERTIFICATE REQUEST----- и оканчивается на -----END CERTIFICATE REQUEST----- (обязательно включая эти два маркера) в отдельный текстовый файл. Дайте ему расширение csr (например partner.csr).

Второй кусок текста, который начинается с текста -----BEGIN PRIVATE KEY----- и заканчивается текстом -----END PRIVATE KEY----- (также включая эти два маркера) в отдельный текстовый файл. Дайте ему расширение key (например partner.key).

Первый файл (partner.csr) это и есть собственно запрос на подпись, а второй (partner.key) — ваш закрытый ключ. Он представляет собой секрет, который не должен разглашаться. В случае компрометации или утери закрытого ключа — скомпрометированный сертификат следует отозвать, и выпустить новый сертификат.

Формирование CSR локально (docker образ)

Более надежным способом (достаточным для промышленного применения) является запуск генератора CSR запросов на локальной машине в docker образе. Это исключит возможную утечку закрытого ключа.

```
$ docker run -d -p 8080:80 --name csrgenerator wittman/csrgenerator.com
```

После запуска открывайте в браузере адрес http://localhost:8080 и проделывайте все шаги по генерации CSR, указанные выше.

Отправка запроса на подпись в Банк

После формирования запроса на подпись (CSR) его следует любым способом отправить его на обработку в Банк (по электронной почте менеджеру в виде файла или открытым текстом). Ваш файл запроса (файл *.csr) не является секретом, поэтому не нуждается в какой либо защите. Закрытый ключ (файл *.key) остается у Партнера и не должен никогда покидать информационную систему Партнера. Он составляет секрет и в случае его компрометации следует отзывать скомпрометированный сертификат и выпускать новый сертификат.

После процедуры проверки и формирования сертификата Партнера вам будет выслан по электронной почте zip файл, который содержит ваш сертификат (в виде файла partner.crt) и корневой сертификат Банка (в виде файла berekebank.crt). Оба файла зашифрованы достаточно длинным паролем. Пароль придет в виде СМС на номер телефона, который вы указывали в заявке на подключение.

Извлеките оба файла из архива и надежно сохраните все три файла (закрытый ключ (partner.key), сертификат Партнера (partner.crt) и корневой сертификат Банка (berekebank.crt)). Эти три файла: всё что нужно для работы Партнера с информационными системами Банка.

Формирование запросов в Банк на стороне Партнера

Для доступа Партнеров к тестовой и продуктовой средам Банком используются различные и отдельные IP адреса и DNS имена:

Контур	Адрес доступа к ИС Банка	Тестовые URI
тестовый	https://b2b.berekebank.kz	https://b2b.berekebank.kz/alive-html https://b2b.berekebank.kz/alive-json
продуктовый	https://authb2b.berekebank.kz	https://authb2b.berekebank.kz/alive-html https://authb2b.berekebank.kz/alive-json

Для формирования доступа к информационной системе Банка с использованием mTLS необходимо соблюсти условие: использование в запросе закрытого ключа и сертификата Партнера.

Формирование сертификата в формате PKCS

Созданный закрытый ключ и полученный сертификат представлены в так называемом формате X509. В некоторых случаях (использование в качестве клиента программы Java или Web браузера) программное обеспечение требует сертификата в ином формате — PKCS12. Обратите внимание что curl и Postman не требуют хранилища PKCS12.

С помощью утилиты OpenSSL формирование сертификата PKCS производится на стороне Партнера самостоятельно (так как PKCS12 хранилище будет содержать и закрытый ключ). Для этого запустите утилиту OpenSSL со следующими параметрами:

```
openssl pkcs12 -export -in partner.crt -inkey partner.key -out partner.p12 -name client -CAfile
berekebank.crt -caname root
```

Где partner.crt — выданный Банком ваш сертификат, partner.key — ваш закрытый ключ, partner.p12 — будущее хранилище ключа и сертификата в формате PKCS, berekebank.crt — корневой сертификат Банка (присланный Банком вместе с вашим сертификатом). После запуска OpenSSL попросит дважды ввести пароль для хранилища сертификатов PKCS. Придумайте и надежно сохраните пароль — без пароля хранилище бесполезно и его придется создавать снова. Созданный утилитой файл partner.p12 может быть использован в Web-браузере напрямую либо в среде Java как хранилище для импорта в Java специфический формат ключей JKS (Java Key Store).

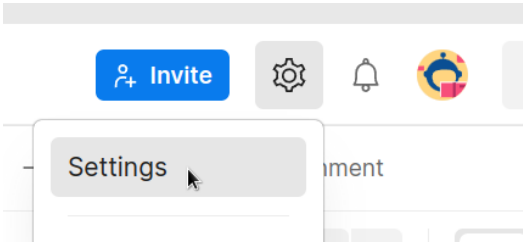
Запросы посредством curl

Программа curl позволяет указывать SSL сертификат при запросе. Делается это через указание параметра --key после которого следует имя файла вашего закрытого ключа. А также указание параметра --cert после которого следует имя файла вашего сертификата, выданного Банком. Корневой сертификат указывается с помощью параметра --cacert. Пример запроса в тестовую среду посредством curl в среде Linux представлен ниже.

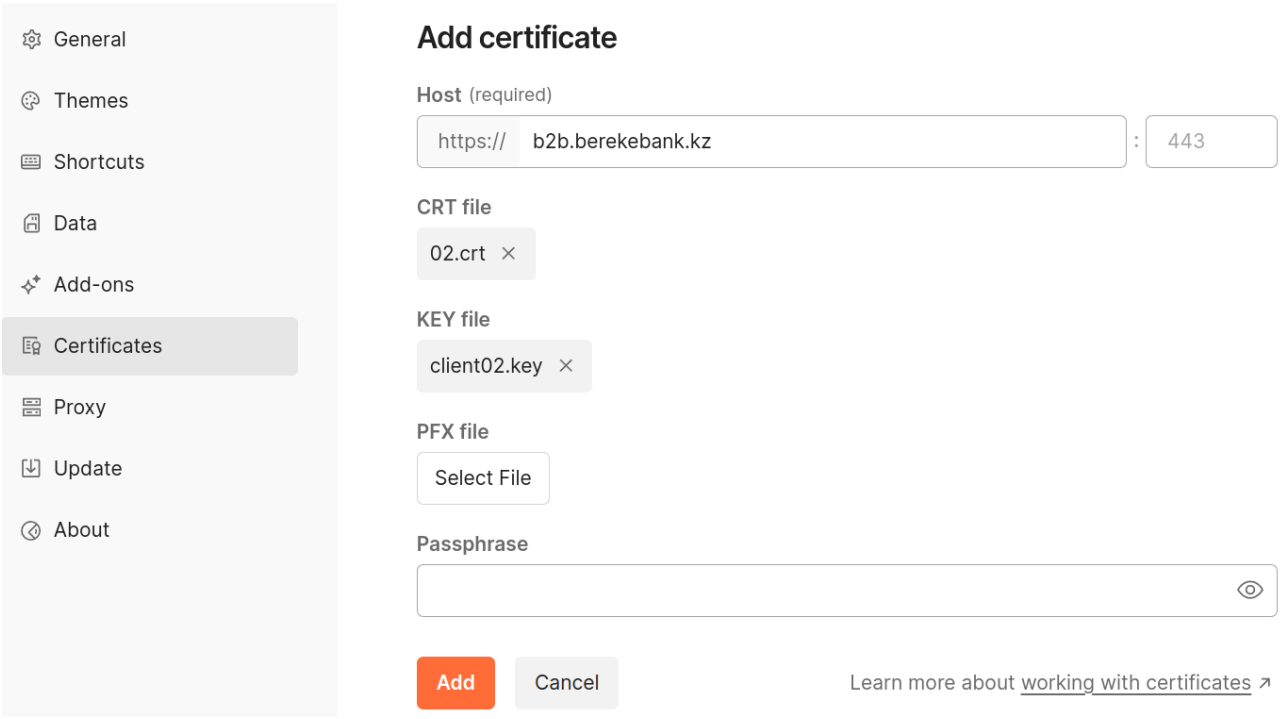
```
max@x670e:/hdd/work/b2b-docs$ curl --key ./partner.key --cert ./partner.crt --cacert ./berekebank.crt -X POST
-H "Content-Type: application/json" -d '{"orderId": "8ec7f5fc-1b5d-77f4-be87-e39b00b98fed"}' https://b2b.berek
ebank.kz/travel/api/v1/status
{"errorCode":"0","errorMessage":"Успешно","orderStatus":2,"cardAuthInfo":""}
max@x670e:/hdd/work/b2b-docs$
```

Запросы посредством Postman

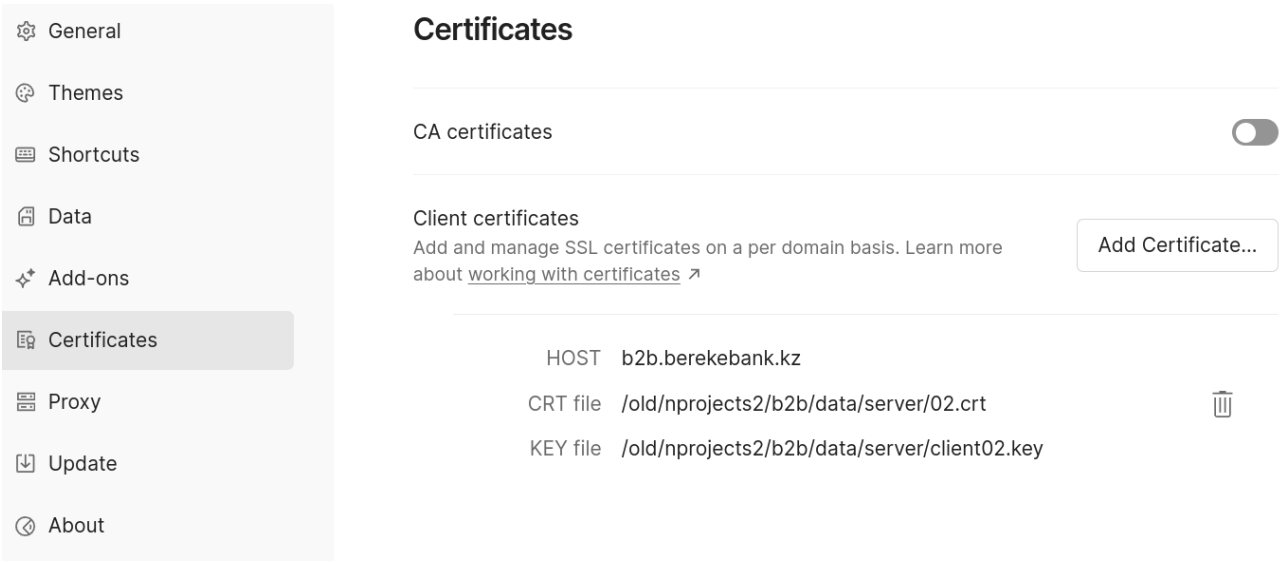
Postman также способен формировать запросы с предъявлением SSL сертификата. Для этого следует указать (для каждого контура свои) настройки в виде адреса сервера, закрытый ключ и сертификат, которые будут применяться при запросе на этот адрес. Настройки находятся в пункте меню.



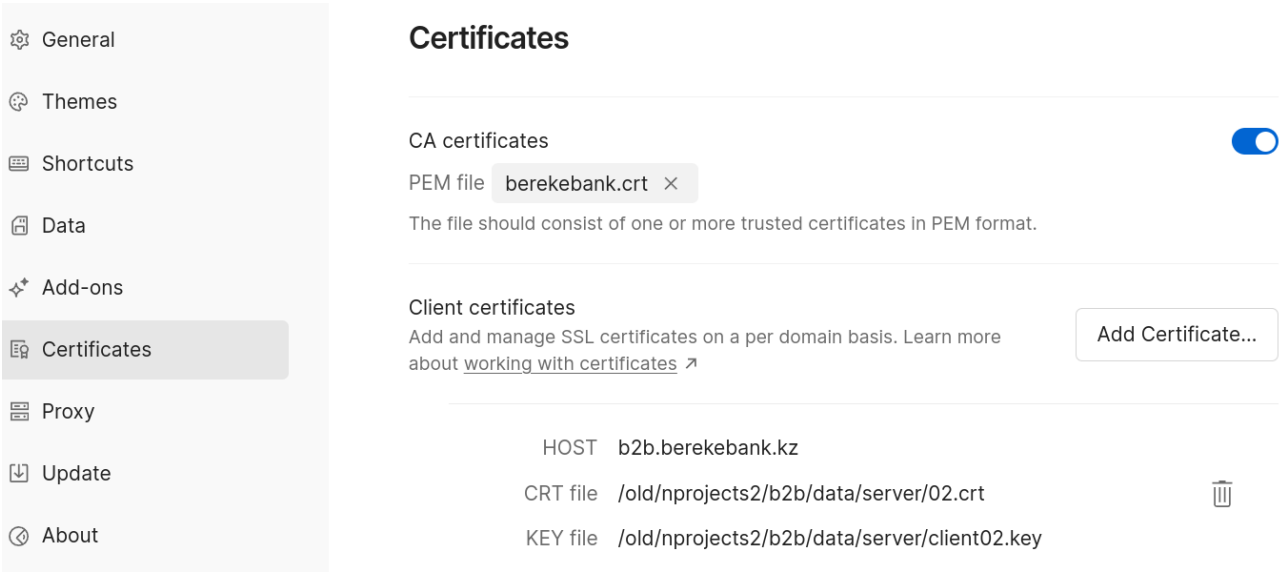
Далее во вкладке «Сертификаты» укажите ваши настройки для тестового контура отдельно и для продуктового отдельно (Add certificate).



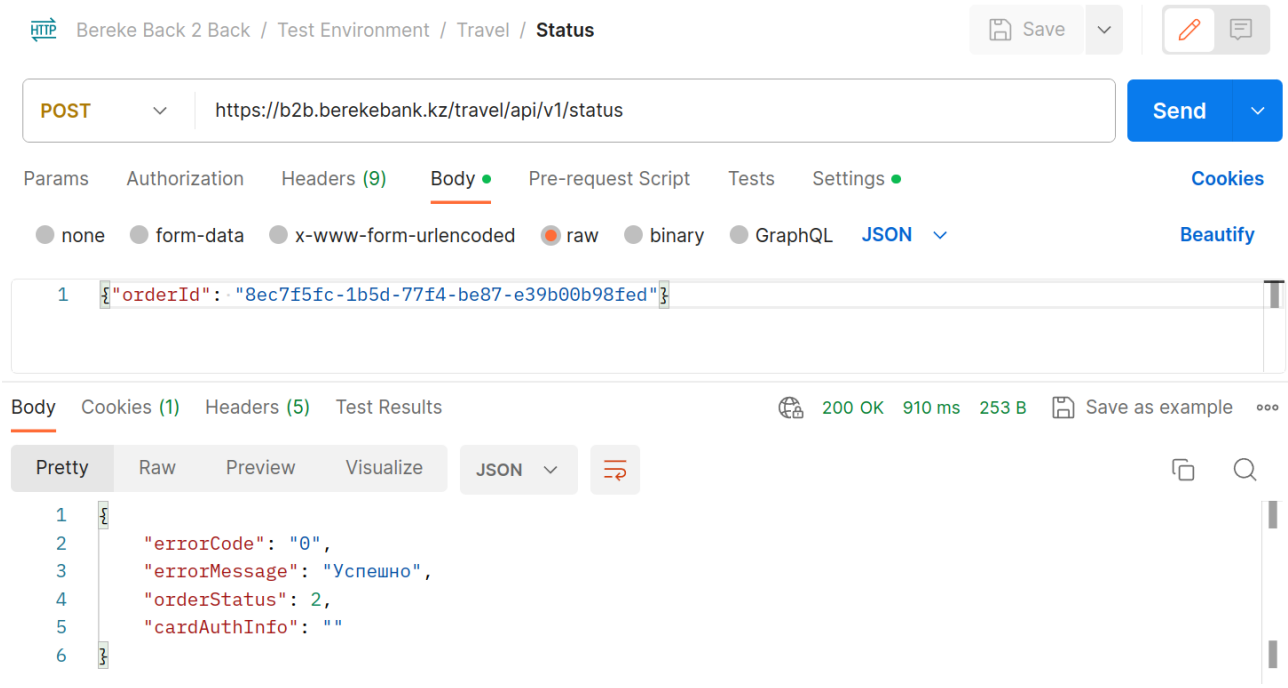
Получившаяся связка для каждого контура содержит адрес контура, ваш закрытый ключ и ваш сертификат.



На этой же вкладке вы можете указать корневой сертификат Банка. Однако Postman позволяет работать как с проверкой корневого сертификата, так и без проверки.



После указания сертификатов Postman обслуживает запросы с аутентификацией сертификатом SSL как обычные HTTP запросы.



Запросы посредством Java (ver >= 11)

Платформа Java обладает собственной реализацией хранилища ключей и сертификатов. Поэтому прежде чем формировать запросы на Java необходимо подготовить Java хранилища с сертификатом Партнера и сертификатом Банка соответственно. Производится это утилитой keytool из JDK. Предполагается работа с Java версии 11 и выше.

Первым создадим хранилище с корневым сертификатом Банка. Для это запустите keytool (из состава JDK) со следующими параметрами:

```
keytool -importcert -keystore server.jks -file berekebank.crt
```

Где server.jks наименование файла JKS который будет создан как хранилище сертификатов, а berekebank.crt — корневой сертификат Банка, полученный вами вместе с вашим сертификатом. После запуска утилиты будет дважды спрошен пароль для хранилища. Придумайте и запишите пароль для хранилища. Без пароля этот файл будет бесполезен и его придется создавать заново. В итоге — созданный файл server.jks будет содержать корневой сертификат Банка.

Далее создадим хранилище сертификатов и ключей Партнера. Напрямую нет способа импортировать закрытый ключ в JKS хранилище. Однако можно импортировать всё содержимое из PKCS хранилища (файл partner.p12 создаваемый выше). Таким образом будут перенесены закрытый ключ и сертификат Партнера. Для этого запустите keytool (из состава JDK) со следующими параметрами:

```
keytool -importkeystore -deststorepass 123456 -destkeypass 123456 -destkeystore client.jks -
srckeystore partner.p12 -srcstoretype PKCS12 -srcstorepass 123456 -alias partner
```

Где deststorepass задаст пароль к хранилищу JKS с сертификатом Партнера, destkeypass задаст пароль для закрытого ключа, client.jks определит имя файла — хранилища JKS для сертификата Партнера, partner.p12 — имя файла хранилища PKCS (созданного выше), srcstorepass — пароль от хранилища PKCS, alias — позволит хранить несколько ключей в одном хранилище под разными псевдонимами.

В результате запуска должен создаться файл client.jks с вашим паролем, закрытым ключом и сертификатом. Созданные два хранилища теперь можно использовать для создания запросов из Java. Также созданные хранилища можно использовать многократно с разными версиями Java на разных системах.

Далее приводится пример на Java с использованием Java Native HTTP Client.

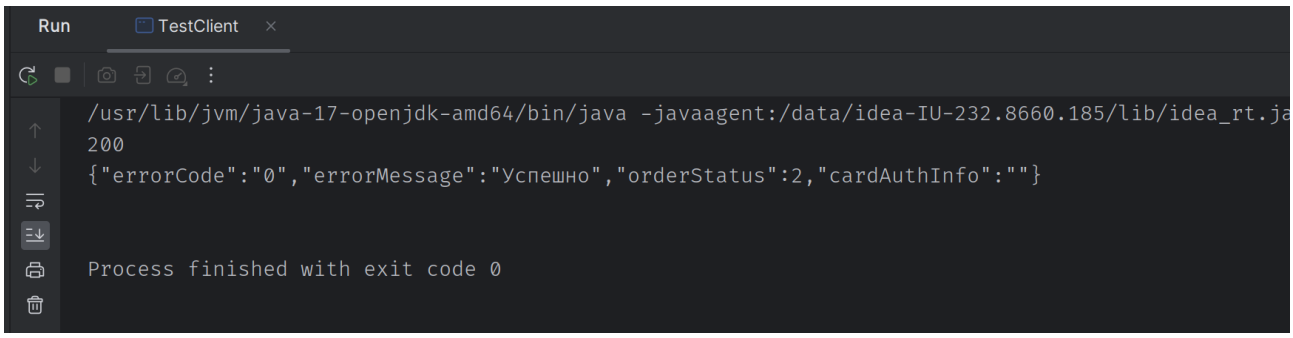
```
package b2b.client;

import javax.net.ssl.KeyManager;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.TrustManager;
import javax.net.ssl.TrustManagerFactory;

import java.io.FileInputStream;
import java.io.IOException;
import java.security.GeneralSecurityException;
import java.security.KeyStore;

public class SSLContextFactory {

    public SSLContext createAndGetSSLContext(String keyStore, String trustStore,
                                             String keyStorePassword, String
trustStorePassword) throws IOException, GeneralSecurityException {
        final KeyManager[] keyManagers = getKeyManagers(keyStore, keyStorePassword);
        final TrustManager[] trustManagers = getTrustManagers(trustStore, trustStorePassword);
        final SSLContext sslContext = SSLContext.getInstance("SSL");
        sslContext.init(keyManagers, trustManagers, null);
        return sslContext;
    }

    private KeyManager[] getKeyManagers(String keyStore, String keyStorePassword) throws
IOException,
            GeneralSecurityException {
        String alg = KeyManagerFactory.getDefaultAlgorithm();
        KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance(alg);
        FileInputStream fis = new FileInputStream(keyStore);
```

```
        KeyStore ks = KeyStore.getInstance("jks");
        ks.load(fis, keyStorePassword.toCharArray());
        fis.close();
        keyManagerFactory.init(ks, keyStorePassword.toCharArray());
        return keyManagerFactory.getKeyManagers();
    }

    private TrustManager[] getTrustManagers(String keyStore, String keyStorePassword) throws
IOException,
            GeneralSecurityException {
        String alg = TrustManagerFactory.getDefaultAlgorithm();
        TrustManagerFactory trustManagerFactory = TrustManagerFactory.getInstance(alg);
        FileInputStream fis = new FileInputStream(keyStore);
        KeyStore ks = KeyStore.getInstance("jks");
        ks.load(fis, keyStorePassword.toCharArray());
        fis.close();
        trustManagerFactory.init(ks);
        return trustManagerFactory.getTrustManagers();
    }

}
```

И собственно пример формирования запроса:

```
package b2b.client;

import javax.net.ssl.SSLContext;
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.security.GeneralSecurityException;
import java.time.Duration;

public class TestClient {

    public static void main(String args[]) throws GeneralSecurityException, IOException,
URISyntaxException, InterruptedException {
        final String keyStorePath = "./data/client.jks"; // путь к JKS сертификата Партнера
        final String keyStorePassword = "123456"; // пароль к JKS сертификата Партнера
        final String trustStorePath = "./data/server.jks"; // путь к JKS Банка
        final String trustStorePassword = "123456"; // пароль к JKS Банка

        final SSLContextFactory sslContextFactory = new SSLContextFactory();
        final SSLContext sslContext = sslContextFactory.createAndGetSSLContext(keyStorePath,
trustStorePath, keyStorePassword, trustStorePassword);

        final HttpClient httpClient = HttpClient.newBuilder()
                .sslContext(sslContext)
                .connectTimeout(Duration.ofSeconds(10))
                .build();

        var req = HttpRequest.newBuilder().
                uri(new URI("https://b2b.berekebank.kz/travel/api/v1/status")).
                header("Content-Type", "application/json").
                POST(HttpRequest.BodyPublishers.ofString("{\"orderId\": \"8ec7f5fc-1b5d-77f4-
be87-e39b00b98fed\"}")).
                build();
        var resp = httpClient.send(req, HttpResponse.BodyHandlers.ofString());
        System.out.println(resp.statusCode());
        System.out.println(resp.body());
    }
}
```

Запуск примера:

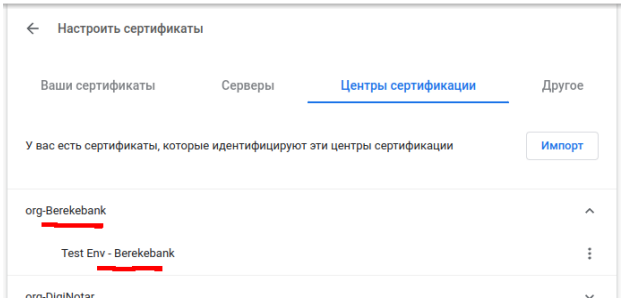


Запросы посредством Web браузера (Chromium)

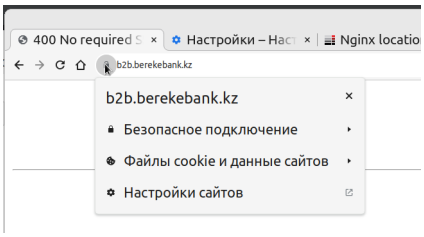
Если в качестве клиента для формирования запросов используется web браузер — то следует учесть, что браузеры часто используют своё, обособленное от системного хранилище корневых сертификатов.

Корневой сертификат в Chrome - Chromium

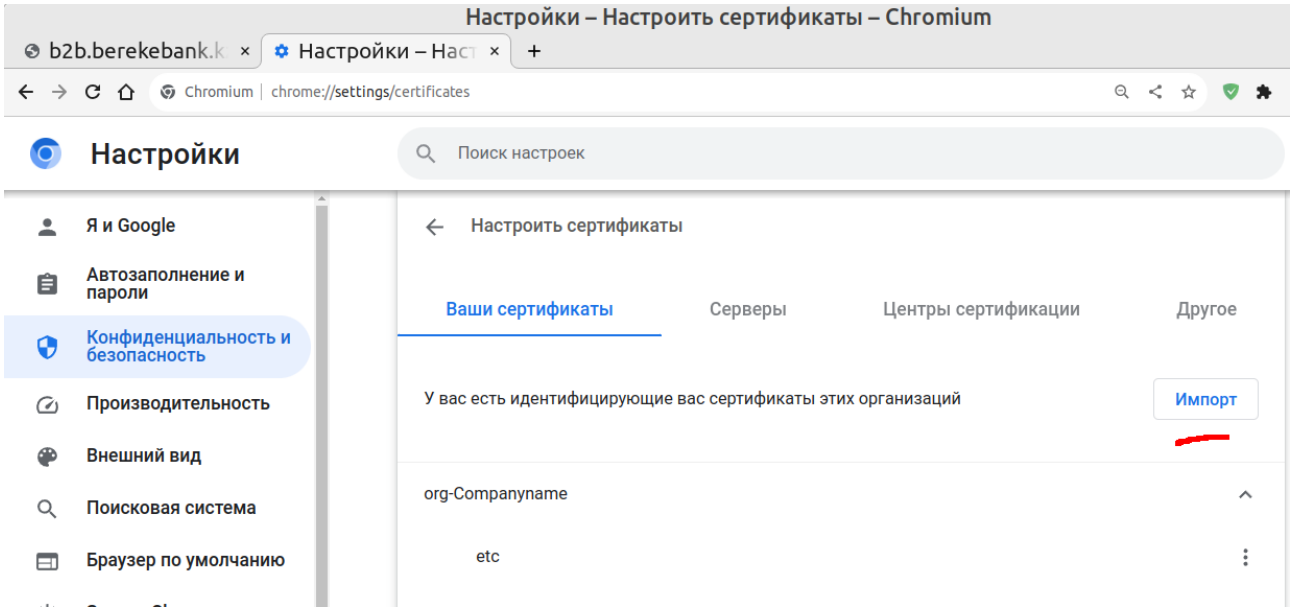
Откройте в меню браузера пункт «Настройки», «Конфиденциальность и безопасность», «Безопасность», «Управление сертификатами устройства». Далее вкладка «Центры сертификации» и кнопка «Импорт». Импортируйте файл berekebank.crt и проследите чтобы он появился в списке центров сертификации.



После этого откройте в этом браузере адрес https://b2b.berekebank.kz для тестового контура или https://authb2b.berekebank.kz для продуктового. Браузер должен распознать сертификат сайта как действующий (значок замочка не будет красным или перечеркнутым).



Чтобы браузер использовал сертификат для доступа к Банку необходимо импортировать ваш сертификат в формате PKCS12 в список ваших личных сертификатов. Создание сертификата в формате PKCS12 описано выше. Импорт сертификата в браузер производится так же как импорт корневого — но на вкладке «Ваши сертификаты».



Теперь вы можете открыть специальную ссылку для проверки параметров своего сертификата: https://b2b.berekebank.kz/alive-html (для тестового контура). Она покажет серийный номер, DN, дату создания и дату годности сертификата, который используется для формирования запроса.