OAuth at Interactive Brokers

November 9, 2017

1 Consumer Registration

Consumers will need to provide the following in order to register as an authorized oauth consumer with Interactive Brokers.

1. A 2048-bit RSA public key for signing HTTP requests (using RSA-SHA256)
2. A 2048-bit RSA public key for encrypting the `access_token_secret`
3. A 2048-bit Diffie-Hellman prime and generator
4. A signature method (currently only RSA-SHA256 is supported)
5. A callback URL to be used during the verification step

Consumers will be issued with a `consumer_key`, a randomly-generated 25-character hexadecimal string. This should be used throughout the oauth process.

A common method to generate the first three parameters is to use `openssl`:

```
openssl dhparam -outform PEM 2048 -out dhparam.pem
openssl genrsa -out private_signature.pem 2048
openssl genrsa -out private_encryption.pem 2048
openssl rsa -in private_signature.pem -outform PEM -pubout -out public_signature.pem
openssl rsa -in private_encryption.pem -outform PEM -pubout -out public_encryption.pem
```

The first command to produce `dhparam.pem` may take several minutes; the rest should complete quickly. After the commands complete, you will have five files:

```
dhparam.pem
private_encryption.pem
private_signature.pem
public_encryption.pem
public_signature.pem
```

You should send Interactive Brokers the files `public_encryption.pem`, `public_signature.pem`, and `dhparam.pem`. The files `private_encryption.pem` and `private_signature.pem` contain your private RSA keys and should not be shared with anyone.

The private keys generated above are in PKCS#1 format. Depending on your platform, it may be easier for you to digest PKCS#8-formatted private keys. You can generate them via the following commands:

```
openssl pkcs8 -topk8 -inform PEM -outform PEM -in private_encryption.pem -out \
   private_encryption.pk8 -nocrypt
openssl pkcs8 -topk8 -inform PEM -outform PEM -in private_signature.pem -out \
   private_signature.pk8 -nocrypt
```

Again, these private keys should not be shared with anyone.

2 Authorization Flow Summary

The goal of the authorization flow is to establish automatically-expiring live session tokens without requiring user re-authorization and without expensive RSA encryption/decryption steps to establish this token. These tokens will then be used to sign subsequent requests (e.g. to place orders) using an HMAC variant.

Toward this goal, two pieces of data will be established between Interactive Brokers and the consumer.

1. An `access_token_secret` (obtained once per user)
2. A Diffie-Hellman shared secret (obtained once per session/day)

The first will be encrypted and transmitted as part of the oauth flow in step 6.3.2 of the oauth 1.0a spec (hereafter "the oauth spec"). Please note that this secret will not be used to sign requests as in section 7 and 9.4.1 of the oauth spec. Instead it will be used by both the consumer and Interactive Brokers in conjunction with the Diffie-Hellman secret produced in §5.3 to compute a "live session token". This live session token will then be used to sign subsequent requests.

3 Messaging format

The only content types supported are `application/w-www-form-urlencoded` and `application/json`.

The oauth spec specifies three possible formats for the oauth protocol parameters; Authorization headers, POST request bodies, or as part of the URL. We will use the first method, Authorization headers.

Responses will be in JSON format.

4 Establishment of `access_token_secret`

We will follow the oauth spec, utilizing the signature method specified during registration.

4.1 Request Token (oauth 6.1)

Request tokens may be obtained by POSTing to the `/oauth/request_token` endpoint with parameters outlined in section 6.1.1 of the oauth spec.

The `oauth_callback` parameter MUST be set to '`oob`'. The callback established as part of the registration procedure in §1 will be supplied to the user during the authentication stage.

Note that we will not return an `oauth_token_secret` in this step as we are using RSA signatures rather than PLAINTEXT authentication.

4.2 User Authorization (oauth 6.2)

After the consumer obtains a request token, the user should be directed to the `/authorize` endpoint. The `oauth_token` in section 6.2.1 is a REQUIRED parameter of this request.

Upon successful authorization, the user will be directed to the callback specified during registration together with the token and verification code in the query string. If authorization is cancelled by the user, they will be redirected to the callback url with no token or verification code specified.

An optional parameter `redirect_uri` may be specified. If present, the path from the callback url specified in §1 is replaced with the value from the parameter.

For example if the registered callback is `https://www.example.com:1234/registration/oauth/v1`, and the consumer directs the user to `/authorize?oauth_token=12af&redirect_uri=/oauth/v2beta`, then after authorization is complete, the user will be directed to

```
https://www.example.com:1234/oauth/v2beta?oauth_token=...&oauth_verifier=...
```

4.3 Access Token (oauth 6.3)

Upon successful receipt of the verification code, the consumer should access the `/oauth/access_token` endpoint using the parameters from section 6.3.1.

A successful request will result in a response with the following parameters:

- `is_paper`
- `oauth_token`
- `oauth_token_secret`

The value of `oauth_token_secret` is the base64-encoded ciphertext of `access_token_secret` mentioned above. It will be enciphered using the public key in step (2) of the registration section §1.

The value of `is_paper` is a boolean indicating whether the user authorized a live trading account (`is_paper=false`) or a paper trading account (`is_paper=true`).

Please note that this `access_token_secret` should never be transmitted to anyone, including Interactive Brokers. We suggest that this token be stored as ciphertext in the consumer's database and be decrypted as needed.

5 Establishment of a `live_session_token`

This step, which is not defined in oauth, establishes a shared secret that will be used to sign requests to access protected resources. The creation of such a key allows us to forego expensive RSA signing/validation of messages used to access protected resources.

Once an `access_token_secret` is obtained, it may be used to obtain a live session token by POSTing to the `/oauth/live_session_token` endpoint. The request should be signed as for steps 6.1-6.3, using the same key and with the slight variation of prepending the `access_token_secret` to the request before computing the signature.

Since RSA signatures require octet strings, we must be explicit about the concatenation method. The deciphered `access_token_secret` is simply a sequence of bytes. This should be converted to a hex representation and concatenated with the signature base string as UTF-8 text. The concatenated UTF-8 string should then be converted into bytes using UTF-8 encoding.

5.1 Request Fields

These fields must be included in the HTTP POST request.

- `oauth_signature`
- `oauth_signature_method`
- `oauth_timestamp`
- `oauth_token`
- `oauth_nonce`
- `diffie_hellman_challenge`

5.2 Response Fields

These fields will be included in the response.

- `diffie_hellman_response`
- `live_session_token_signature`

5.3 Diffie-Hellman challenge/response

The consumer will select an integer a and compute $A = g^a \bmod p$ where g and p are the generator and prime from the registration step. The integer A should be sent in hex representation in the `diffie_hellman_challenge` field of the request. After validating the request, Interactive Brokers will select an integer b and compute $B = g^b \bmod p$, which will be encoded in hex format in the response field `diffie_hellman_response`.

Our shared secret is then $K = A^b \bmod p = B^a \bmod p$. The live session token is then computed as

`live_session_token = HMAC_SHA1(K, access_token_secret)`.

We will then compute `HMAC_SHA1(live_session_token, consumer_key)` and transmit it in the response as the `live_session_token_signature` parameter. This will allow the consumer to verify the correct calculation of `live_session_token`.

6 Accessing protected resources

A live session token remains valid for 24 hours after its creation. All subsequent requests should signed using the live session token as the key in an HMAC scheme. The particulars of the request format and signing (e.g. via Json Web Tokens, oauth-style signature base strings, etc.) are specific to the protected resource itself.

6.1 OAuth `HMAC_SHA256` signatures

Currently the only supported method for request signing is OAuth-style `HMAC_SHA256` signatures. We require the Authorization header to be present and the first token of that header to be `OAuth` followed by the required by §7 of the oauth spec.

To summarize the requirements in that section, the following components of the `Authorization` header are required

- `OAuth`
- `oauth_consumer_key`
- `oauth_nonce`
- `oauth_token`
- `oauth_signature`
- `oauth_signature_method`
- `oauth_timestamp`
- `realm`

The `oauth_signature_method` must be `HMAC_SHA256`, and `oauth_token` is the access token obtained in §4.3. For `POST` and `PUT` requests, the `Content-Type` header must be `www-form-urlencoded` and the body must match that type. The signature base string is constructed exactly as in the oauth spec §9. Please note that the sorting of parameters is lexicographic and case-sensitive. The signature base string is signed using the live session token as the key in the `HMAC_SHA256`:

`HMAC_SHA256`(Live Session Token, Signature Base String).

See §7.6 for an explicit example.

7 Example

7.1 Registration

The consumer registers with Interactive Brokers, supplying Interactive Brokers with two pairs of 2048-bit RSA keys, a signature method, and a Diffie-Hellman prime and group as in §1. See §8 for the examples used in this section. The consumer specifies `RSA-SHA256` as the encryption method, and supplies `https://www.example.com/callback` as the callback. The consumer is then issued with a consumer key:

$$\texttt{consumer\_key} = \texttt{TESTCONS}.$$

In this example, we have utilized a proxy on `localhost:12345` to make explicit the Host header and signature base string construction.

7.2 Obtaining a request token

The consumer makes a POST request through the proxy to

`https://www.interactivebrokers.com/tradingapi/v1/oauth/request_token`

(the '\' character indicates line continuation in the following).

7.2.1 HTTP request

```
POST /tradingapi/v1/oauth/request_token HTTP/1.1
Authorization: OAuth oauth_callback="oob", \
    oauth_consumer_key="TESTCONS", \
    oauth_nonce="fcbc9c08d69ac269f7f1", \
    oauth_signature="nHp...AA%3D%3D", \
    oauth_signature_method="RSA-SHA256", \
    oauth_timestamp="1473793701", \
    realm="test_realm"
Content-Length: 0
Host: localhost:12345
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.1 (Java/1.8.0_102)
Accept-Encoding: gzip,deflate
```

7.2.2 Signature Base String

The signature base string for this request is

```
POST&http%3A%2F%2Flocalhost%3A12345%2Ftradingapi%2Fv1%2Foauth%2Frequest_token&oaut
h_callback%3Doob%26oauth_consumer_key%3DTESTCONS%26oauth_nonce%3Dfcbc9c08d69ac269f7
f1%26oauth_signature_method%3DRSA-SHA256%26oauth_timestamp%3D1473793701
```

7.2.3 Signature

The full signature is

```
nHpiG+/oGBtXtQPYOEc1PJKVUYCgFzbbUyQ1slb5GtvRVp+nfOKlm4eXt7P4DhNecsrg5gEc/AWHD+L7sn
XFmOWlj6c1XjcSCj1r8ERWHg/f8U43PwLkVHCarwgh3cVcHOKGioPpShefNcVvriAUxfBiScWKiB/1zmI1
cs9yBHqoJS6pv4KlAGeLj3eXLmYphZGgJzLMLyj6X4UIT1RftHLKbqwmgfVMoBuDm5EDJDokN2VowepIMn
kQBjiUJGWklUcgD8Jq9VHAgzTfdMsX33Nnfia+Z4ZcdJLUn2uUlNJX3FtWJeYO2OImvXoQOZNXN6hiUH7q
```

158Z/LOqWDyUAA==

7.2.4 HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 38
x-response-time: 79ms

{"oauth_token":"25ebcc75204da80b73f4"}
```

7.3 Access Token

We assume that the user has logged in and obtained a verification code `61c107d4cf34ac6d9f2b` for the request token `25ebcc75204da80b73f4`.

7.3.1 HTTP request

```
POST /tradingapi/v1/oauth/access_token HTTP/1.1
Authorization: OAuth oauth_consumer_key="TESTCONS", \
    oauth_nonce="afd6f94d3784db186f0e", \
    oauth_signature="YbO1...rWBA%3D%3D", \
    oauth_signature_method="RSA-SHA256", \
    oauth_timestamp="1473793702", \
    oauth_token="25ebcc75204da80b73f4", \
    oauth_verifier="61c107d4cf34ac6d9f2b"
Content-Length: 0
Host: localhost:12345
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.1 (Java/1.8.0_102)
Accept-Encoding: gzip,deflate
```

7.3.2 Signature Base String

The signature base string for this request is

```
POST&http%3A%2F%2Flocalhost%3A12345%2Ftradingapi%2Fv1%2Foauth%2Faccess_token&oauth_
consumer_key%3DTESTCONS%26oauth_nonce%3Dafd6f94d3784db186f0e%26oauth_signature_meth
od%3DRSA-SHA256%26oauth_timestamp%3D1473793702%26oauth_token%3D25ebcc75204da80b73f4
%26oauth_verifier%3D61c107d4cf34ac6d9f2b
```

7.3.3 Signature

The full signature is

```
YbO1qrhY8HvK7+wUg7q8pxyBv4IeqcCqvZkES+XrCGvP77KZP7D+7iovvLjnyyOMlzuNpqNtZEVmDzN/9O
GYwEAZWAhTMESlqB9+kmrBpg51/vIhlFUiZBvjLcqNyH49aKpbJvlCliSs3eHi9GWoHdEb1jDzu01UK3vs
iFrSLfYTVc7xqviq3Ml5TBgwb8Ccxk28PFYRu4W9NSlez9J/jBkkwTwEpmeIn6RT1zTypXv5gyVCzeJdgJ
g1U4LzHIa26w+Tli0EmqKPLdwWwW57wQaOzegBeKqWmwrqbmDzbXoFz7FiXl4NXgcGXwB/yWjMgChhE0Vj
iBgfk3W1zTrWBA==
```

7.3.4 HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 422
x-response-time: 42ms

{"is_paper":true,\
 "oauth_token":"6f531f8fd316915af53f",\
 "oauth_token_secret":"MtUT...GJA=="}
```

The full secret is

```
MtUTi9TxkzPUTS4UTMb64ALEIK4tAsrXg/fsUuyc6WmEj5lV5oHXRGGpc3zpNkRIDHeCm+iwQuvuDl7mxi
LZihHDpIF4huU7flWloyTKN4JObe5TBKnzm2FQpOSUqrzho87dVSD1N1CDhHiV4suE3E1lMayYQKYsAOsI
JcnI63ZZeIc0UyBtEE4cUhwgmgAE5Ag2atZx4CRk5mqFgxd1ayRQiHc+v+r+B9DCK8vLe9uT5KPbHDEN35
XVt3RRiyUxtQQFc9qaYfJvY/ZCXqnp7DSUrUAOJKNyOfrn7SRG3ME/HmhAZC/Q79Pr9qSGSjJIbFBlUQYG
GhsNpkdDIjcGJA==
```

This secret is encrypted using the public key in §8.2 and can be decrypted using the private key in §8.3 to obtain the access token secret

```
R2bzBq10CLvaoZUM9PM3EBVV0PpCq5BIceL+V+NlsnI=
```

7.4 Establishment of a Live Session Token

The consumer chooses an integer

$$a = 1435329019564828319111943272230435123117133842132517761382825 \quad (1)$$

and compute $A = g^a \bmod p$, the full hex representation of which is

```
adcc3e6d1a297418336fd90f41f0b1a1d9b025b35725f6803d6b13309bc3d0fcfdaeff17306bcafa5d
0e91a66ad540254cacae28550e30145df9d7a3847bb7774b6c53a6f1e5c1aaed51fffb17807c8e2083
d93ede25801b41a83dd9fcce5b3f8cff4200dff23ebf907c6eab820a35fc32133eb09c653d7ceebbad
f14715a3c191a37a442d1063232ddbc7fbc1be855d62b7383e134175e33c19b9118d6e3213e5996641
87319b39960efc5eb7e9f0e891d3bc71fd7e0f13f0330c0edf8f67007e5bf327219569298bea3ebde9
c772c2b9461f484ed956e888c7c545f11a05c02812ef07ea026d0bd69a0b2fe60d7c106e059515a088
780ebd1143b0765bebb
```

The value of A is then encoded the '`diffie_hellman_challenge`' field of the request.

7.4.1 HTTP request

```
POST /tradingapi/v1/oauth/live_session_token HTTP/1.1
Authorization: OAuth diffie_hellman_challenge="adcc...bebb", \
    oauth_consumer_key="TESTCONS", \
    oauth_nonce="36f7d85e418f8bfe8561", \
    oauth_signature="KzaCo...eBUw%3D%3D", \
    oauth_signature_method="RSA-SHA256", \
    oauth_timestamp="1473793702", \
    oauth_token="6f531f8fd316915af53f"
Content-Length: 0
Host: localhost:12345
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.1 (Java/1.8.0_102)
Accept-Encoding: gzip,deflate
```

7.4.2 Signature Base String

We prepend the secret `R2bzBq10CLvaoZUM9PM3EBVV0PpCq5BIceL+V+NlsnI=` in hex when constructing the signature base string.

```
4766f306ad7408bbdaa1950cf4f337101555d0fa42ab904871e2fe57e365b272POST&http%3A%2F%2F
localhost%3A12345%2Ftradingapi%2Fv1%2Foauth%2Flive_session_token&diffie_hellman_cha
llenge%3Dadcc3e6d1a297418336fd90f41f0b1a1d9b025b35725f6803d6b13309bc3d0fcfdaeff173
06bcafa5d0e91a66ad540254cacae28550e30145df9d7a3847bb7774b6c53a6f1e5c1aaed51fffb178
07c8e2083d93ede25801b41a83dd9fcce5b3f8cff4200dff23ebf907c6eab820a35fc32133eb09c653
d7ceebbadf14715a3c191a37a442d1063232ddbc7fbc1be855d62b7383e134175e33c19b9118d6e321
3e599664187319b39960efc5eb7e9f0e891d3bc71fd7e0f13f0330c0edf8f67007e5bf327219569298
bea3ebde9c772c2b9461f484ed956e888c7c545f11a05c02812ef07ea026d0bd69a0b2fe60d7c106e0
59515a088780ebd1143b0765bebb%26oauth_consumer_key%3DTESTCONS%26oauth_nonce%3D36f7d8
5e418f8bfe8561%26oauth_signature_method%3DRSA-SHA256%26oauth_timestamp%3D1473793702
%26oauth_token%3D6f531f8fd316915af53f
```

7.4.3 Signature

The full signature is

```
KzaCo+vE5k1T3aor2/th/IhS1gj1b81cLTPVZYtsHsMYoyn0xY2f7Gkxnc9i1ZgeM5JFdh5qz32GhDlArb
hHodSRxKEGPJ2BqOGlEBsL1RymJYY0sujoMg+uw06Ti7pg7h6OWdYi+yskRxbstD4O1Z4ZZXn6s/gBrO60
+V5ZQe7ENsgbg7pvHmzjREC4NSrWLA4MQIbJimzZJ04JChvrSRsTcopnRdj3OXg9ce2eGFgPW1cbCV4fNo
Bg51M8BHOxGB10yux9usRBfh6UdzCaTHSbskhxIrUGLYc04SMh0KNE99dxww2w5vnyGPk2QnnPyRTcNBAM
5UjerB3CqFeBUw==
```

7.4.4 HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 615
x-response-time: 77ms

{"diffie_hellman_response":"51...0e","live_session_token_signature":"54...f4"}
```

Here, the full Diffie-Hellman response is

```
51a8175da12f6952e935756321e0d3589cbd7c2535413deafe67ea4395da94af3ac589ec99d0680cbf
29f475c56450fd66bc403080bedc8be1805408e461acd6ae0f00fbb3ee8e81927448edff8b011af0f7
2eeefb4d2bc1ae8099d1e62cb9d2963e84195c8dce1e43d0694d32be7651f108d9b973439da6690b8f
9409ffcba6f5e588e05611d161edb09464babd10fa84310d62551775745cacb5bb5071f179181eda83
aea6de7bdba997ed8820a52cf7c84d41605895bdddb44972f06726866cd30472a8f53c1d50ba4d92c7
9737a1f54ffba404b389ee8c14ed1082140347658413781 1acbfca733147db6b776af4261af7cf9ff3
224a3043ec705af760e
```

and the full live session token signature is `543c55477d6cbb0e792d1e4f8111cec7305ba3f4`. The Diffie-Hellman integral value is referred to as B below.

7.4.5 Live Session Token Calculation

Using a from equation (1), and B from §7.4.4,

$$\begin{aligned} K &= B^a \bmod p \\ &= 626246873 \ldots 1995994781 \end{aligned}$$

The full decimal representation of K is

```
6262468731191716683916525460981637432269697150114366475089957235034556054645865209
7727466193918128491278351377747537675165331206606577881303560193892583906396297713
0285681861128917558036055166314200723585833452614598893039983793075173417203185616
8887029288967531594210251515114214270648331076558038219524053473452048253881320127
1884787556741085253161782311759691397680487566604132871880546809466098442097571479
8594795590947008746401693222302341952432123212575873140979848617085540126903548680
6152769695999321982943785989707170257093535822989962797048573685671123477713856278
6624160048492915383636497157993199599478l
```

K is converted to its big-endian byte representation and used to compute the live session token as:

$$\texttt{LST = HMAC\_SHA1(K, secret) = YBWbLw+9RYP2nWrPQHxHZkBb1aM=}$$

where `secret` is `R2bzBq10CLvaoZUM9PM3EBVV0PpCq5BIceL+V+NlsnI=`, the decrypted secret bytes.

One can use the `live_session_token_signature` field to verify that the live session token has been correctly computed. One computes

$$\texttt{HMAC\_SHA1(LST, consumer\_key\_bytes) = 543c55477d6cbb0e792d1e4f8111cec7305ba3f4.}$$

Here, the signature is HEX encoded, and `consumer_key_bytes` is the `UTF-8` decoded bytes of the string `TESTCONS`.

7.5 Accessing Protected GET Resources

We give an example of accessing a protected resource that utilizes `HMAC_SHA256` signatures with the live session token.

7.5.1 HTTP Request

```
GET /tradingapi/v1/marketdata/snapshot?conid=8314 HTTP/1.1
Authorization: OAuth realm="test_realm",\
    oauth_consumer_key="TESTCONS",\
    oauth_signature="+BdI...v8%3D",\
    oauth_signature_method="HMAC-SHA256",\
    oauth_nonce="aecef17086308940e861",\
    oauth_timestamp="1473795686",\
    oauth_token="6f531f8fd316915af53f"
Host: localhost:12345
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.1 (Java/1.8.0_102)
Accept-Encoding: gzip,deflate
```

7.5.2 Signature Base String

The signature base string is

```
GET&http%3A%2F%2Flocalhost%3A12345%2Ftradingapi%2Fv1%2Fmarketdata%2Fsnapshot&conid
%3D8314%26oauth_consumer_key%3DTESTCONS%26oauth_nonce%3Daecef17086308940e861%26oaut
h_signature_method%3DHMAC-SHA256%26oauth_timestamp%3D1473795686%26oauth_token%3D6f5
31f8fd316915af53f
```

7.5.3 Signature

The UTF-8 encoded bytes of the signature base string are signed as

```
HMAC_SHA256(LST, SBS) = +BdIuZDNooYZAbO9RZUCTC5F/3HjFOb04Tu4crpi0v8=
```

7.5.4 HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 146
x-response-time: 64ms

{"Closing":{"price":158.29},"Trade":{"price":155.76,"size":1,"time":1473795686},\
"Bid":{"price":155.74,"size":2},"Offer":{"price":155.76,"size":5}}
```

7.6 Accessing Protected POST Resources

Here we give an example of accessing a protected POST resource. Again we will utilize HMAC_SHA256 signatures. Suppose that we have obtained a new live session token hsSvwnDjYhhMj3Ub2wKmMCCenMQ= and have visited the /accounts endpoint to discover that the access token 6f531f8fd316915af53f as an associated account DU216409.

7.6.1 HTTP Request

```
POST /ptradingapi/v1/accounts/DU216409/order_impact HTTP/1.1
Authorization: OAuth oauth_consumer_key="TESTCONS",\
    oauth_nonce="fafd0982f8db1e34287c",\
    oauth_signature="PsRc%2F99DBX4AyZyWqHnUJrEhsf2tTn%2BUWg6gafI01us%3D",\
    oauth_signature_method="HMAC-SHA256",\
    oauth_timestamp="1475766474",\
    oauth_token="6f531f8fd316915af53f",\
    realm="test_realm"
Content-Length: 115
Content-Type: application/x-www-form-urlencoded; charset=ISO-8859-1
Host: localhost:12345
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.5.1 (Java/1.8.0_102)
Accept-Encoding: gzip,deflate

CustomerOrderId=ibm1&ContractId=8314&Exchange=SMART&\
Quantity=100&Price=100&OrderType=Limit&TimeInForce=DAY&Side=BUY
```

7.6.2 Signature Base String

The signature base string is

```
POST&http%3A%2F%2Flocalhost%3A12345%2Fptradingapi%2Fv1%2Faccounts%2FDU216409%2Ford
er_impact&ContractId%3D8314%26CustomerOrderId%3Dibm1%26Exchange%3DSMART%26OrderTyp
e%3DLimit%26Price%3D100%26Quantity%3D100%26Side%3DBUY%26TimeInForce%3DDAY%26oauth_
consumer_key%3DTESTCONS%26oauth_nonce%3Dfafd0982f8db1e34287c%26oauth_signature_meth
od%3DHMAC-SHA256%26oauth_timestamp%3D1475766474%26oauth_token%3D6f531f8fd316915af53
f
```

7.6.3 Signature

The UTF-8 encoded bytes of the signature base string are signed as

```
HMAC_SHA256(LST, SBS) = PsRc/99DBX4AyZyWqHnUJrEhsf2tTn+UWg6gafI01us=
```

7.6.4 HTTP Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 259
x-response-time: 98ms

{"EquityWithLoan":9899985.0,\
 "EquityWithLoanBefore":9899985.0,\
 "InitMargin":14504.36,\
 "InitMarginBefore":10578.76,\
 "MaintMargin":14504.36,\
 "MaintMarginBefore":10578.76,\
 "MarginCurrency":"USD",\
 "MinCommissions":1.0,\
 "MaxCommissions":2.35,\
 "CommissionsCurrency":"USD"}
```

8 Appendix

This section contains Base64-encoded values used in the example in §7. Please note that the encrption keys below are in PKCS#8 format. You may obtain the corresponding public keys via `openssl`; the encryption public key is given explicition for verification purposes.

8.1 Signing Key

```
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYwggSiAgEAAoIBAQDDgozsh3e7
rl59JI+OPYXJyc0Vlftmm45tTxXQ6PlzYsQifuOAwfK0bc2HPvMlVH/sxYEU
GR6H4K3D9j//uFbm6dg1/4YFTg68tnoReUk6x/kz8gF/8WKB7cVysSQvhWPK
+zGyDXq4w7J4XMpV9gSfeuIqkXchCwvkTVbysGB6C7jwySeiTRL/XhQfNj6i
GnLos60Uz7IuY0zuvJGvJIdgLTnI61MCaz59DzXhk0ATyZR00hfgGvvjNxBN
OUVvTh8wq18NWer207che8iYXF0kHTEO8jabWv9k/6Ol+NBW2YnKD3WcEcda
/sii+CAH+YmXVcoOdtCj9oJKp96voUQrAgMBAAECggEAAYlOGDG8VCI7pdoa
srBzE7HdRbUnA5hnVx6O+lRKWYjGJ8s9eG6NvLM8MTriaEGwu6iVqhXEcqbm
Vqrht923+RqPU2VNpU8GuJ7qeKauV6Lop2rYbiv0bTb0HUGdGi5/KUIuEsse
UsNe7Kk2AdTsFmC91D3OZNkUX9fnvgJ6z5qyc1SvZGbrUiJuOMzXFz7g0/YH
31H1mK72PIpFp5rha1djWzi0Pc99v06HwURIJAy/aB7aaB9oXWKf0xw94+PG
yUUUbfFJ0n3mgx7PQcOEaebu0JsGLBb28AUbughjjD9A9qcIjJUbdwWq6MYY
1JZfvymVuEurAaCLozdEWb0f2QKBgQDyaAdzlZL1SCNv9Jg+sBm8u1fkH1pR
WtZ6iQZLE46H6nwvOL4ZMm2Va/6SK6Jn7vikLHrXhizOanJuvgKjgYKiAknj
X8NGgTfObgh8X92xGch6m9irLLJoPHSBfAegr1Aro/z8rJuqdcBzyiHz7xof
ZZkr/B1WLE96A/g9tVxTLQKBgQDOeUeOuY8B2iYVY0EZQQ4Yriyk/n1fgz3T
JFc8OuKJiir6WnYRSueC5DBsI0ghDQLq4VCG8yAkIUk/w98lGDaGGw+c+Mn1
mAvAbTN7p/Rkv5h1EEJqkWL8EyVbQj3yMlKVXz0fJ05ShD7c1FwV4pasiY5y
Yy6UK1Lkmr3DRhZrtwKBgAm9Jjd9lebm3A2ebRNvrckIWdKfc9h65s0LfkY7
5ekOAalABVsrzH1JVnP27tJwJmsqIlFQYbJ1Ul2ioqaBCIeLwf4xlb3aaSyQ
/SRKEbUQzR4n7r+Jzy0i6KdfHUtWX8kxEHPyV1q02mOMhmqve4QxZ8L9oQwa
QaMsT6fjBm+NAoGAY4LsTAt+syPSqwg1Pdiu1JJNjg/hmQEF8RdYu/ydZjCj
WhnzQY5aWilkdRnkD5nXyCBj0UaoPQGV7heXGqe6z5OHSN5XZ/ip4UpAP97I
2S3GatU1TWtYy6jmnj8k3/AFAVzvdm5ZP0fhjEkFQL7+Y5XAg7ztnBftUaSK
xOyora0CgYACDDuIJjJ7Xym49iIzJcs+w4KSLpDdvXt5yVuJoOhMeLgsA0JQ
dqUbBrEo6tTzdDMeUWuEwqfC4la5myQojkzyjZSFrnhk9ecBYxNwoDPDVL7/
sviL3SEWu87gRfyIW9AtUZgr2iibyL9KXNcRc5D1kaiQj1DbJ5XOsRycWvhh
XQ==
-----END PRIVATE KEY-----
```

8.2 Encryption Public Key

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqXXw7Xv7G7VQDBNiWrOd
oKj9RaBV/6Xorx0zJy8iqTh10YBoVpA04mGLQWnOPRi7XKmH4smfsmWb7zWbNEup
PtEttbpczNx8Q4akM9w6CGBBiKkVPhOFTaxg/8AAWFWpNCiSkFDnI9goh9AfMy3p
PkKxUIjlfDKnboMA7842uuIvGfVrkyEvQz5w46fl1J8l3Qm4V/3SBU4T9w+RBT50
gqZ5zr+8aev61gfgfMMX7XkQT8Rxn70ybIlyt+ICXfR2raFTExi4h0HjCC5jRmyH
ZW5eEft6FKXOw+6okzUVNr4k9gcoumBIMnpP5Y7Gy1Yj+aSkLp52Jtywz82pmF2o
yQIDAQAB
-----END PUBLIC KEY-----
```

8.3 Encryption Private Key

```
-----BEGIN PRIVATE KEY-----
MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQCpdfDte/sb
tVAME2Jas52gqP1FoFX/peivHTMnLyKpOHXRgGhWkDTiYYtBac49GLtcqYfi
yZ+yZZvvNZs0S6k+0S21ulzM3HxDhqQz3DoIYEGIqRU+E4VNrGD/wABYVak0
KJKQUOcj2CiH0B8zLek+QrFQiOV8MqdugwDvzja64i8Z9WuTIS9DPnDjp+XU
nyXdCbhX/dIFThP3D5EFPnSCpnnOv7xp6/rWB+B8wxfteRBPxHGfvTJsiXK3
4gJd9HatoVMTGLiHQeMILmNGbIdlbl4R+3oUpc7D7qiTNRU2viT2Byi6YEgy
ek/ljsbLViP5pKQunnYm3LDPzamYXajJAgMBAAECggEAfY4aBQsAISZ0eY3/
KG6+UmQCy4yhSzn0rZI37UdaKgDgImo+ZCySUkytbbjkCpaBc4VUXe6/OOFZ
6KFSLiwE6Lr5YBUFXu/Zzfo7PQVlpgETbRrB122FLeiTMcpjNw+4vyF0fiLD
LuagGQhVpYjVsMCfOalERoINVa8QiZzDJjDo/MoNuIv3+SGKpVfwI+imQH0U
FbeYRQ26fMZ7blDineajiTfTvR74pRZzfyGRlr5iquc6Y6MkpvZnhoBziRUH
mte59gM25hLh8cKdRJc8zPKIQcmEFi3CeCuV9O/I3o1LtFl9qkvBA9i6vSK9
S70de8zvNZ/wqxs4T1JvnHjHAQKBgQDzpkPY7ianRPaLonUkz44uQczJ6ihh
vIUxClX2nD49tblsmN5Gh2naQ70i5ZLPx6XkJ1P90kSLK7LPRpPRrb4jHEBM
K/MCtK+yNSr/pRXL1UpsIZPXqfhP37jANf0ujogccZUrwy6RbagvqYYllkA6
O3604oPo6ZD5eAWxgzcP4QKBgQCyDPXRdCekbkZF22J1i2AiDptY7B/nEAJG
t/ouQhycONh7jVle30Me9pADfFkEFn5cyWt2nmCrNPMNg3kwjxmgYOpmNYL9
rjRzo+Hr4J+B5oiQJOqrUq9ZGdwXMEuy73qri3d8py0u+zhhrFav3IbtAFvl
K/rNHzPxHMAKRErV6QKBgQDJOP59sSmRKX5okfInRZdKEqO7+jwBg/2II94c
/qrUsjmCO7oPC5TbYuhYYrteCKe0BtryHG7pdVyGnOZF3DTUsGdVwVJAYNNd
3VU2rrOSlQOSzCRaj5B4/u6JJ82CXsqAmzm0W80g5CxruJkGf9Fphms9nnmM
qSFHNt2U3iZiYQKBgHzf8hfwq0z7unWMJEXkdovsSq/XC5j29WG77s86tu4q
QEqHHEMZuu2gZ+jJ8XSYgg95hNEwywPIox67MugHtSVyU3NT3IcnAsqICg9w
/u4QTX8rJRr7ZtfK9z2cNbtNkm915VubtGja8PQv9qxosojOU6Urbz8lvzw/
ucB5nTLhAoGBAJdt9TkorV1z8FFO9inSOQPu8GFGqG312UFCbKVXsA8cRrN0
QriIkubkXH+f6nlb4VenLcbtPj+WFfDeQthDPpjnQ/N3Y2KtfIetZoFmFyJr
fOrodzjCiPf/1aPqdjo84Ya9iTD9i9YsREgBJIF1VJ3JYvFekdjTTrHbCC8k
+Fdf
-----END PRIVATE KEY-----
```

8.4 Diffie-Hellman Parameters

Prime:

```
24990697953968689185706675227002407939775030447575733097395112952082357221424093747693551272240612471853311757548349330811169427040859224263719226162247345178775079328454444354273614727946229069542432045539665380486423225027487524348368337927704203276897027870335481135977094556035258459224370209596331597955215983129372140107533724032882879952343559495540859849703806286204331571698018713450924441193021081900668422143031817823935003777420923589915333926271708107190834366387204345935981036234722096758651171238449058638798183975712160444662670167115295419507567659220491095
```

74203376555104517793742187938294533592428 5

Generator:
20987177111196158644366843622345789285608866677846058262388416761283779644758807784668755785792948410551747724872422109659421471903405265108031859454328782621600226092159632991563737358469987305806274484811391639032843020172237818843190174302921872566141394330154841222190528859792032158789434556396592945071874891845382615101191695518809097477517288420096235237460819485746068277784085816072231788200368891178303122154313607460176772166332845852125160120267796594522794546036187528669075504174889390839605502026020485104344329151603365980900821301873348531910373702447838936174629657554031816697007791113193297648393