WebSocket API General Info

- The base endpoint is: `wss://ws-fapi.binance.com/ws-fapi/v1`
  - The base endpoint for testnet is: `wss://testnet.binancefuture.com/ws-fapi/v1`
- A single connection to the API is only valid for 24 hours; expect to be disconnected after the 24-hour mark.
- Websocket server will send a ping frame every 3 minutes.
  - If the websocket server does not receive a `pong frame` back from the connection within a 10 minute period, the connection will be disconnected.
  - When you receive a ping, you must send a pong with a copy of ping's payload as soon as possible.
  - Unsolicited pong frames are allowed, but will not prevent disconnection. **It is recommended that the payload for these pong frames are empty.**
- Signature payload must be generated by taking all request params except for the signature and sorting them by name in alphabetical order.
- Lists are returned in **chronological order**, unless noted otherwise.
- All timestamps are in **milliseconds in UTC**, unless noted otherwise.
- All field names and values are **case-sensitive**, unless noted otherwise.
- `INT` **parameters such as timestamp are expected as JSON integers, not strings.**
- `DECIMAL` **parameters such as price are expected as JSON strings, not floats.**
- **User Data Stream requests - you will need to establish a separate WebSocket connection to listen to user data streams**

WebSocket API Request format

Requests must be sent as JSON in **text frames**, one request per frame.

> Example of request:

```
{
  "id": "9ca10e58-7452-467e-9454-f669bb9c764e",
  "method": "order.place",
  "params": {
    "apiKey": "yeqKcXjtA9Eu4Tr3nJk61UJAGzXsEmFqqfVterxpMpR4peNfqE7Zl7oans8Qj089",
    "price": "42088.0",
    "quantity": "0.1",
    "recvWindow": 5000,
    "side": "BUY",
```

```
    "signature": "996962a19802b5a09d7bc6ab1524227894533322a2f8a1f8934991689cabf8fe",
    "symbol": "BTCUSDT",
    "timeInForce": "GTC",
    "timestamp": 1705311512994,
    "type": "LIMIT"
  }
}
```

Request fields:

Name	Type	Mandatory	Description
`id`	INT/STRING/null	YES	Arbitrary ID used to match responses to requests
`method`	STRING	YES	Request method name
`params`	OBJECT	NO	Request parameters. May be omitted if there are no parameters

- Request `id` is truly arbitrary. You can use UUIDs, sequential IDs, current timestamp, etc. The server does not interpret `id` in any way, simply echoing it back in the response.

You can freely reuse IDs within a session. However, be careful to not send more than one request at a time with the same ID, since otherwise it might be impossible to tell the responses apart.

- Request method names may be prefixed with explicit version: e.g., "`v3/order.place`".
- The order of `params` is not significant.

Response format

Responses are returned as JSON in text frames, one response per frame.

> Example of successful response:

```
{
  "id": "43a3843a-2321-4e45-8f79-351e5c354563",
  "status": 200,
```

```
  "result": {
    "orderId": 336829446,
    "symbol": "BTCUSDT",
    "status": "NEW",
    "clientOrderId": "FqEw6cn0vDhrkmfiwLYPeo",
    "price": "42088.00",
    "avgPrice": "0.00",
    "origQty": "0.100",
    "executedQty": "0.000",
    "cumQty": "0.000",
    "cumQuote": "0.00000",
    "timeInForce": "GTC",
    "type": "LIMIT",
    "reduceOnly": false,
    "closePosition": false,
    "side": "BUY",
    "positionSide": "BOTH",
    "stopPrice": "0.00",
    "workingType": "CONTRACT_PRICE",
    "priceProtect": false,
    "origType": "LIMIT",
    "priceMatch": "NONE",
    "selfTradePreventionMode": "NONE",
    "goodTillDate": 0,
    "updateTime": 1705385954229
  },
  "rateLimits": [
    {
      "rateLimitType": "REQUEST_WEIGHT",
      "interval": "MINUTE",
      "intervalNum": 1,
      "limit": 2400,
      "count": 1
    },
    {
      "rateLimitType": "ORDERS",
      "interval": "SECOND",
      "intervalNum": 10,
      "limit": 300,
      "count": 1
    },
    {
      "rateLimitType": "ORDERS",
      "interval": "MINUTE",
      "intervalNum": 1,
      "limit": 1200,
      "count": 0
    }
  ]
}
```


Example of failed response:

```
{
  "id": "5761b939-27b1-4948-ab87-4a372a3f6b72",
  "status": 400,
  "error": {
    "code": -1102,
    "msg": "Mandatory parameter 'quantity' was not sent, was empty/null, or malformed."
  },
  "rateLimits": [
    {
      "rateLimitType": "REQUEST_WEIGHT",
      "interval": "MINUTE",
      "intervalNum": 1,
      "limit": 2400,
      "count": 1
    },
    {
      "rateLimitType": "ORDERS",
      "interval": "SECOND",
      "intervalNum": 10,
      "limit": 300,
      "count": 1
    },
    {
      "rateLimitType": "ORDERS",
      "interval": "MINUTE",
      "intervalNum": 1,
      "limit": 1200,
      "count": 1
    }
  ]
}
```


Response fields:

Name	Type	Mandatory	Description
`id`	INT/STRING/null	YES	Same as in the original request
`status`	INT	YES	Response status. See status codes
`result`	OBJECT/ARRAY	YES	Response content. Present if request succeeded
`error`	OBJECT	YES	Error description. Present if request failed
`rateLimits`	ARRAY	NO	Rate limiting status. See Rate limits

WebSocket API Rate limits

- Rate limits are the same as on REST API and are shared with REST API.
- WebSocket handshake attempt costs 5 weight.
- Rate limit for ping/pong frames: maximum 5 per second.
- Rate limit information is included in responses by default, see the `rateLimits` field.
- `rateLimits` field visibility can be controlled with `returnRateLimits` boolean parameter in connection string or individual requests.
- E.g., use `wss://ws-fapi.binance.com/ws-fapi/v1?returnRateLimits=false` to hide `rateLimits` in responses by default. With that, you can pass extra `"returnRateLimits": true` parameter in requests to show rate limit in response when it is otherwise hidden by default.

WebSocket API Authenticate after connection

You can authenticate an already established connection using session authentication requests:

- `session.logon` - authenticate, or change the API key associated with the connection
- `session.status` - check connection status and the current API key
- `session.logout` - forget the API key associated with the connection

WebSocket API API key revocation

If during an active session the API key becomes invalid for any reason (e.g. IP address is not whitelisted, API key was deleted, API key doesn't have correct permissions, etc), after the next request the session will be revoked with the following error message:

```
{
  "id": null,
  "status": 401,
  "error": {
    "code": -2015,
    "msg": "Invalid API-key, IP, or permissions for action."
  }
}
```

WebSocket API Authorize ad hoc requests

Only one API key can be authenticated with the WebSocket connection. The authenticated API key is used by default for requests that require an apiKey parameter. However, you can always specify the apiKey and signature explicitly for individual requests, overriding the authenticated API key and using a different one to authorize a specific request.

For example, you might want to authenticate your USER_DATA key to be used by default, but specify the TRADE key with an explicit signature when placing orders.

WebSocket API Authentication request

Note:

> Only *Ed25519* keys are supported for this feature.

Log in with API key (SIGNED)

> **Request**

```
{
  "id": "c174a2b1-3f51-4580-b200-8528bd237cb7",
  "method": "session.logon",
  "params": {
    "apiKey": "vmPUZE6mv9SD5VNHk4HlWFsOr6aKE2zvsw0MuIgwCIPy6utIco14y7Ju91duEh8A",
    "signature": "1cf54395b336b0a9727ef27d5d98987962bc47aca6e13fe978612d0adee066ed",
    "timestamp": 1649729878532
  }
}
```

> **Response**

```
{
  "id": "c174a2b1-3f51-4580-b200-8528bd237cb7",
  "status": 200,
  "result": {
    "apiKey": "vmPUZE6mv9SD5VNHk4HlWFsOr6aKE2zvsw0MuIgwCIPy6utIco14y7Ju91duEh8A",
    "authorizedSince": 1649729878532,
    "connectedSince": 1649729873021,
    "returnRateLimits": false,
    "serverTime": 1649729878630
  }
}
```

Authenticate WebSocket connection using the provided API key.

After calling `session.logon`, you can omit `apiKey` and `signature` parameters for future requests that require them.

Note that only one API key can be authenticated. Calling `session.logon` multiple times changes the current authenticated API key.

Weight: 2

Method: "session.logon"

Parameters

Name	Type	Mandatory	Description
`apiKey`	STRING	YES	
`recvWindow`	INT	NO	
`signature`	STRING	YES	
`timestamp`	INT	YES	

Query session status

Request

```
{
  "id": "b50c16cd-62c9-4e29-89e4-37f10111f5bf",
  "method": "session.status"
}
```

Response

```
{
  "id": "b50c16cd-62c9-4e29-89e4-37f10111f5bf",
  "status": 200,
  "result": {
    // if the connection is not authenticated, "apiKey" and "authorizedSince" will be
shown as null
    "apiKey": "vmPUZE6mv9SD5VNHk4HlWFsOr6aKE2zvsw0MuIgwCIPy6utIco14y7Ju91duEh8A",
    "authorizedSince": 1649729878532,
```

```
    "connectedSince": 1649729873021,
    "returnRateLimits": false,
    "serverTime": 1649730611671
  }
}
```

Query the status of the WebSocket connection, inspecting which API key (if any) is used to authorize requests.

Weight: 2

Method: "session.status"

Parameters: None

Log out of the session

Request

```
{
  "id": "c174a2b1-3f51-4580-b200-8528bd237cb7",
  "method": "session.logout"
}
```

Response

```
{
  "id": "c174a2b1-3f51-4580-b200-8528bd237cb7",
  "status": 200,
  "result": {
    "apiKey": null,
    "authorizedSince": null,
    "connectedSince": 1649729873021,
    "returnRateLimits": false,
    "serverTime": 1649730611671
  }
}
```

Forget the API key previously authenticated. If the connection is not authenticated, this request does nothing.

Note that the WebSocket connection stays open after `session.logout` request. You can continue using the connection, but now you will have to explicitly provide the `apiKey` and `signature` parameters where needed.

Weight: 2

Method: "session.logout"

Parameters: None

SIGNED (TRADE and USER_DATA) Endpoint Security

SIGNED request example (Ed25519)

Parameter	Value
symbol	BTCUSDT
side	SELL
type	LIMIT
timeInForce	GTC
quantity	1
price	0.2
timestamp	1668481559918

```
#!/usr/bin/env python3

import base64
import time
import json
from cryptography.hazmat.primitives.serialization import load_pem_private_key
from websocket import create_connection

# Set up authentication
API_KEY='put your own API Key here'
PRIVATE_KEY_PATH='test-prv-key.pem'

# Load the private key.
```

```
# In this example the key is expected to be stored without encryption,
# but we recommend using a strong password for improved security.
with open(PRIVATE_KEY_PATH, 'rb') as f:
    private_key = load_pem_private_key(data=f.read(),
                                       password=None)

# Set up the request parameters
params = {
    'apiKey':        API_KEY,
    'symbol':       'BTCUSDT',
    'side':         'SELL',
    'type':         'LIMIT',
    'timeInForce':  'GTC',
    'quantity':     '1.0000000',
    'price':        '0.20'
}

# Timestamp the request
timestamp = int(time.time() * 1000) # UNIX timestamp in milliseconds
params['timestamp'] = timestamp

# Sign the request
payload = '&'.join([f'{param}={value}' for param, value in sorted(params.items())])

signature = base64.b64encode(private_key.sign(payload.encode('ASCII')))
params['signature'] = signature.decode('ASCII')

# Send the request
request = {
    'id': 'my_new_order',
    'method': 'order.place',
    'params': params
}

ws = create_connection("wss://ws-fapi.binance.com/ws-fapi/v1")
ws.send(json.dumps(request))
result =  ws.recv()
ws.close()

print(result)
```

A sample code in Python to show how to sign the payload with an Ed25519 key is available on the right side.