WebAuthn uses public key certificates instead of passwords to complete user registration and identity authentication (login). It is more like an enhancement or supplement to existing identity authentication. In order to ensure the security of communication data, it is generally based on HTTPS (TLS) communication.

1. Server: It can be considered as a relying party (Relying Party), it will store the user's public key and is responsible for user registration, authentication.
2. Browser: Credential Management API with WebAuthn is required
3. Authentication module (Authenticator): It can create, store, and retrieve identity credentials. It is generally a hardware device (smart • card, USB), or it may have been integrated into your operating system (such as Windows Hello).

1: Register

(a) Initiate a registration request

(b) The server returns Challenge, user information, relying party information

(c) The browser calls the authentication module to generate a certificate

This is an asynchronous task, and the JS script calls the browser's navigator.credentials.create to create a certificate.

```
getMakeCredentialsChallenge({username, name})
        .then((response) => {
            let publicKey = preformatMakeCredReq(response);
            return navigator.credentials.create({ publicKey })
        })
        .then((response) => {
            console.log(response);
            let makeCredResponse = publicKeyCredentialToJSON(response);
            return sendWebAuthnResponse(makeCredResponse)
        })
        .then((response) => {
            if(response.status === 'ok') {
                loadMainContainer()
            } else {
                alert(`Server responed with error. The message is: ${response.messa
            }
        })
```

(d) The authentication module creates a pair of public / private key and attestation data

(e) The authentication module sends the public key / Credential rawID / attestation to the browser

(f) The browser sends Credential to the server

(g) Server completes registration

2: Authentication

Same as most steps for registration

Call navigator.credentials.get of the browser to retrieve the certificate.

```
getGetAssertionChallenge({username})
        .then((response) => {
            console.log(response)
            let publicKey = preformatGetAssertReq(response);
            return navigator.credentials.get({ publicKey })
        })
        .then((response) => {
            console.log(response)
            let getAssertionResponse = publicKeyCredentialToJSON(response);
            return sendWebAuthnResponse(getAssertionResponse)
        })
        .then((response) => {
            if(response.status === 'ok') {
                loadMainContainer()
            } else {
                alert(`Server responed with error. The message is: ${response.messa
            }
        })
```

3: Task

(a) Algorithm:

```
import urllib
import base64
import getpass
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
import base64

import simplejson
import requests
import cbor2 as cbor
from fido2.hid import CtapHidDevice
from fido2.client import Fido2Client
```

```
class Fido2HttpClient(object):
    session = None
    server = None
    dev = None
    ssl_verify = True
    begin_url = None
    complete_url = None
    is_authenticated = False
    verbose = False

    def authenticate_to(self,
            server, begin_endpoint, complete_endpoint,
            session=None, append_to_data=None, append_to_headers=None):
        self.server = server
        self.init_dev()
        begin_url = urllib.parse.urljoin(server, begin_endpoint)
        complete_url = urllib.parse.urljoin(server, complete_endpoint)

        if self.dev:
            if session:
                self.session = session
            if not self.session:
                self.session = requests.session()

            self.begin(begin_url,
                append_to_headers=append_to_headers)
            self.complete(complete_url,
                append_to_data=append_to_data,
                append_to_headers=append_to_headers)
        return self.is_authenticated

    def log(self, *args):
        if self.verbose: print(args)

    def ask_for_interaction(self):
        print('Touch your authenticator device...')

    def no_device_found(self):
        print('No FIDO device found')

    def authenticated(self):
        print('Authenticated')

    def say_not_authenticated(self, data):
        print('Not Authenticated')

    def init_dev(self):
```

```
        self.dev = next(CtapHidDevice.list_devices(), None)
        if not self.dev:
            self.no_device_found()

    def begin(self, begin_url, append_to_headers=None):
        headers = {}
        if append_to_headers:
            for k in append_to_headers:
                headers[k] = append_to_headers[k]

        r = self.session.post(begin_url,
            verify=self.ssl_verify,
            headers=headers)

        self.begin_data = cbor.loads(r.content)
        self.log('BEGIN RESPONSE: ', self.begin_data)

    def complete(self, complete_url, append_to_data=None, append_to_headers=None):
        fido2_client = Fido2Client(self.dev, self.server)

        pubkey = self.begin_data['publicKey']
        challenge = base64.b64encode(pubkey['challenge'])
        challenge = challenge.decode('utf-8')
        allow_list = [{
            'type': 'public-key',
            'id': pubkey['allowCredentials'][0]['id'],
        }]

        self.ask_for_interaction()
        try:
            assertions, client_data = fido2_client.get_assertion(
                pubkey['rpId'],
                challenge,
                allow_list)
        except ValueError:
            assertions, client_data = fido2_client.get_assertion(
                pubkey['rpId'],
                challenge,
                allow_list,
                pin=getpass.getpass('Please enter PIN:'))

        assertion = assertions[0]
        self.log('ASSERTION: ', assertion)
        self.log('CLIENT DATA: ', client_data)

        user_data = {}
        if append_to_data:
            user_data = append_to_data
```

```
        user_data = base64.b64encode(simplejson.dumps(user_data).encode('utf-8'))

        data = {
            'credentialId': assertion.credential['id'],
            'authenticatorData': assertion.auth_data,
            'clientDataJSON': client_data,
            'signature': assertion.signature,
            'user_data': user_data,
        }
        body = cbor.dumps(data)
        headers = {'content-type': 'application/cbor'}
        if append_to_headers:
            for k in append_to_headers:
                headers[k] = append_to_headers[k]

        r = self.session.post(complete_url,
            verify=self.ssl_verify,
            headers=headers,
            data=body
        )
        self._last_response = r

        data = cbor.loads(r.content)
        self.log('Response: ', data)
        if 'status' in data and data['status'] == 'OK':
            self.is_authenticated = True
        else:
            self.is_authenticated = False

        if self.is_authenticated: self.authenticated()
        else: self.say_not_authenticated(data)
        return self.is_authenticated

    def get_last_response(self):
        return self._last_response
```

(b) Output:

```
▲ ~/Desktop/python-fido2-client python test.py                                    master :: 528d :: ○
('BEGIN RESPONSE: ', {'publicKey': {'rpId': 'localhost', 'timeout': 30000, 'challenge': b'\x1d\n\xa0!?  \x8a\xcd\xca\x
1a\xdb\xa2}\xe2\xf7\x9e\x8dyvC{\x83\x08\xa2>o;\x17\x11\x1e\x945\xb3', 'allowCredentials': [{'id': b'\x1e.\xd8c\xcd*\x8
a\xebI!\t\x9d\x9d\x99-\xb6\x7f\xfbf\xf5\xa0\xab\xa4@\xbc\xe9\x0e\xf6\xf2^\xbaG:\x07\xdcef\xef\xcf\x0e\xf6\xda\xa9\xbf\
x06\x84O\xfb\x00e\x88\x7f\xa7\x11\x00g\x90`\xdf\x85\x97\x95Rf', 'type': 'public-key'}], 'userVerification': 'preferred
'}})

Touch your authenticator device...

('ASSERTION: ', AssertionResponse(credential: {'id': b'\x1e.\xd8c\xcd*\x8a\xebI!\t\x9d\x9d\x99-\xb6\x7f\xfbf\xf5\xa0\x
ab\xa4@\xbc\xe9\x0e\xf6\xf2^\xbaG:\x07\xdcef\xef\xcf\x0e\xf6\xda\xa9\xbf\x06\x84O\xfb\x00e\x88\x7f\xa7\x11\x00g\x90`\x
df\x85\x97\x95Rf', 'type': 'public-key'}, auth_data: AuthenticatorData(rp_id_hash: h'49960de5880e8c687434170f6476605b8
fe4aeb9a28632c7995cf3ba831d9763', flags: 0x01, counter: 63), signature: h'3046022100c9080974ae855029e00d2d770ae78cb1f5
24d9953d1f3c5e73e1055ea0ac6a5902210082edf3c9f339e78b3d21ee96bba7b677e7c98542ab4191676cc2f840fa7514b2'))
('CLIENT DATA: ', {"type": "webauthn.get", "clientExtensions": {}, "challenge":   "HQqgIT+Kzcoa26J94veejXl2Q3uDCKI+bzs
XER6UNbM=", "origin": "https://localhost:5000"\})
('COMPLETE RESPONSE: ', {'status': 'OK'})
```