Applying Decentralised Identity Techniques to Web Authentication

Nicole Khor, Anonyome Labs

December 2020

Motivation

- To implement a solution that allows decentralised identity (or SSI) to authenticate with a wider range of services
- Leverage Web Authentication (WebAuthn) due to its existing support in industry
- Create a hybrid approach to SSI based password-less logins
    - ‘Plugs’ into existing services

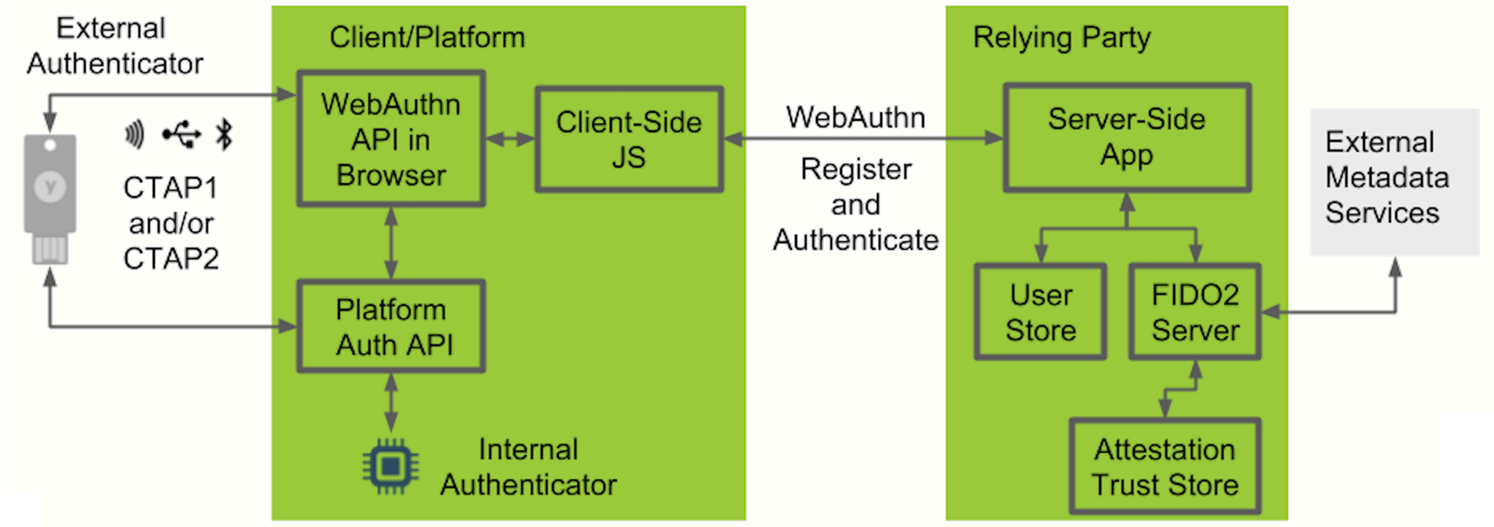
WebAuthn

- The official password-less authentication standard for ”strong, attested, and scoped public key based credentials” [1].
- Allows for the authentication of users into enabled services
- A component of the Fast Identity Online 2 (FIDO2) project
- By the Fast Identity Online (FIDO) Alliance and the World Wide Web Consortium (W3C)

[1] D. Balfanz, A. Czeskis, J. Hodges, J. Jones, M. B. Jones, A. Kumar, R. Lindemann, and E. Lundberg, “Web authentication: An api for accessing public key credentials.” https://www.w3.org/TR/webauthn-2/, 2019

FIDO2 Overall

WebAuthn and Client to Authenticator Protocol (CTAP)

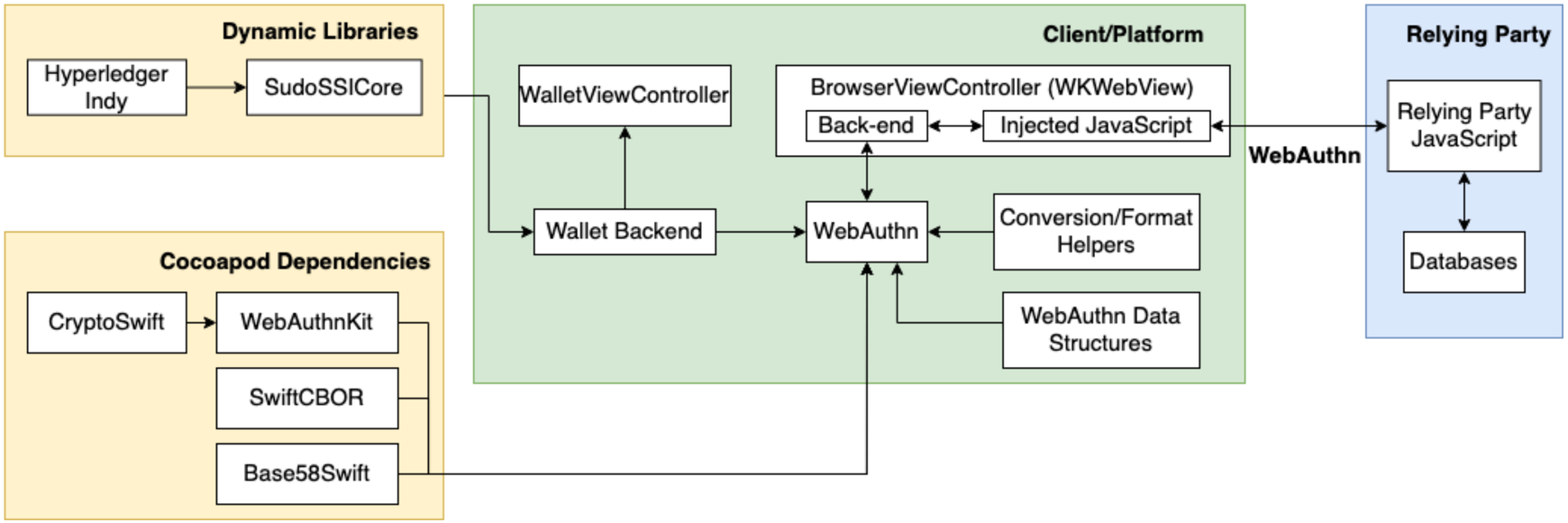


[2]

[2] Yubico, “Fido2/webauthn overview.” https://developers.yubico.com/WebAuthn/ WebAuthn Developer Guide/Overview.html, 2020

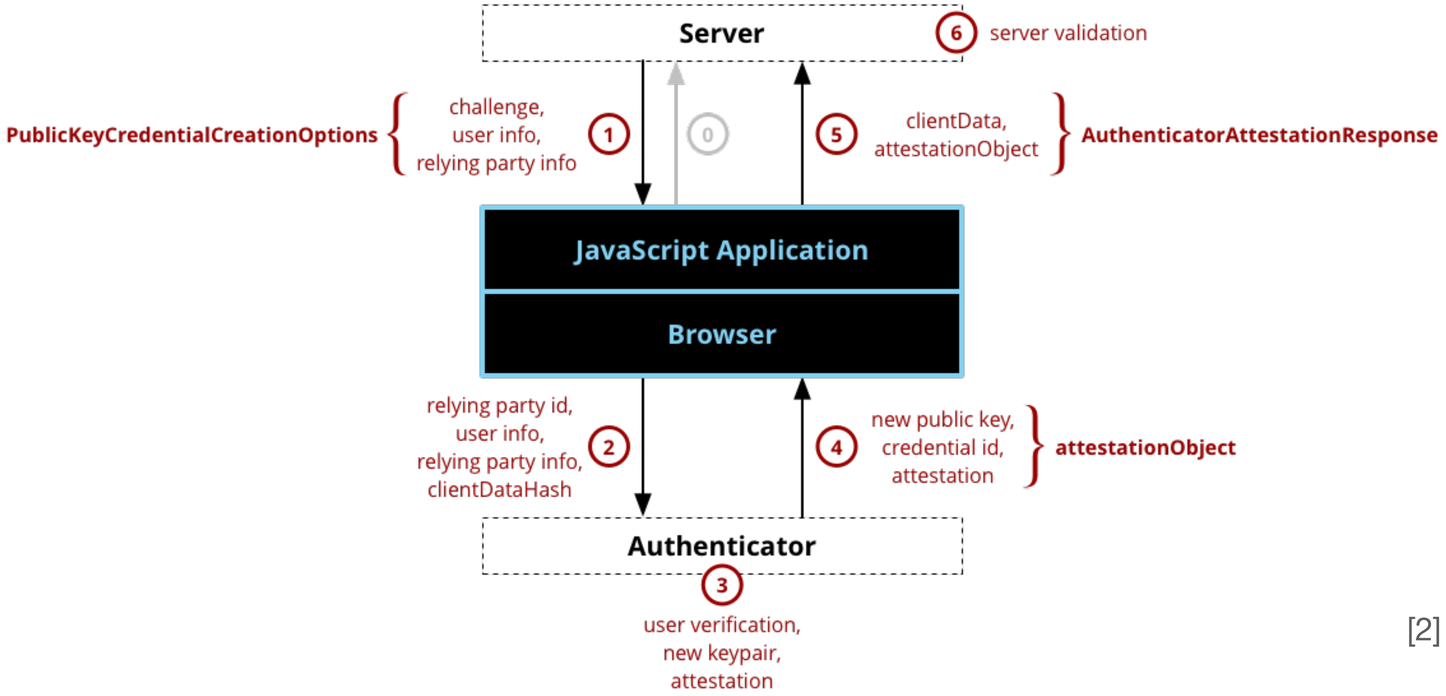
Integrated SSI into WebAuthn

- iOS client in Swift
- Python based relying party from Duo Labs



Registration Ceremony

- Registering an identity
- Providing the credential identifier, public key and attesting the authenticator

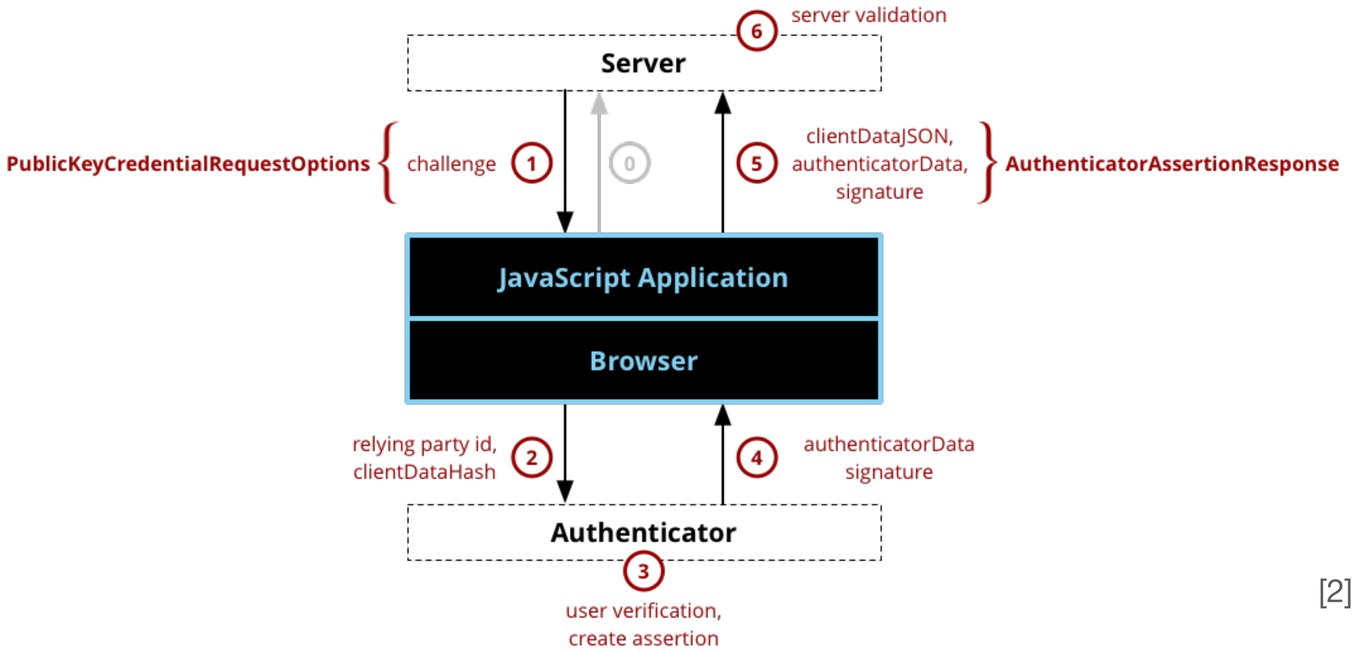


[2]

[2] Yubico, “Fido2/webauthn overview.” https://developers.yubico.com/WebAuthn/ WebAuthn Developer Guide/Overview.html, 2020

Authentication Ceremony

- Signing a challenge using previously given public key

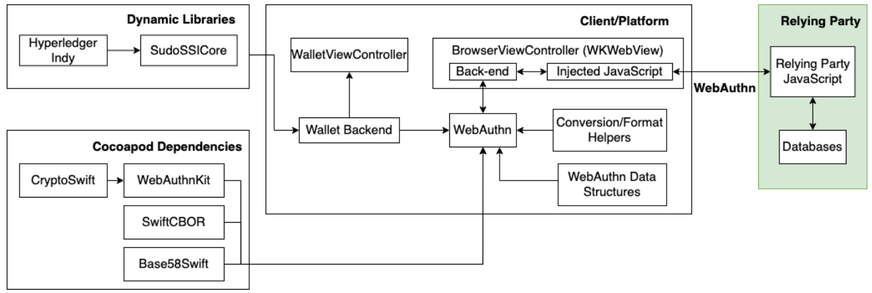


[2]

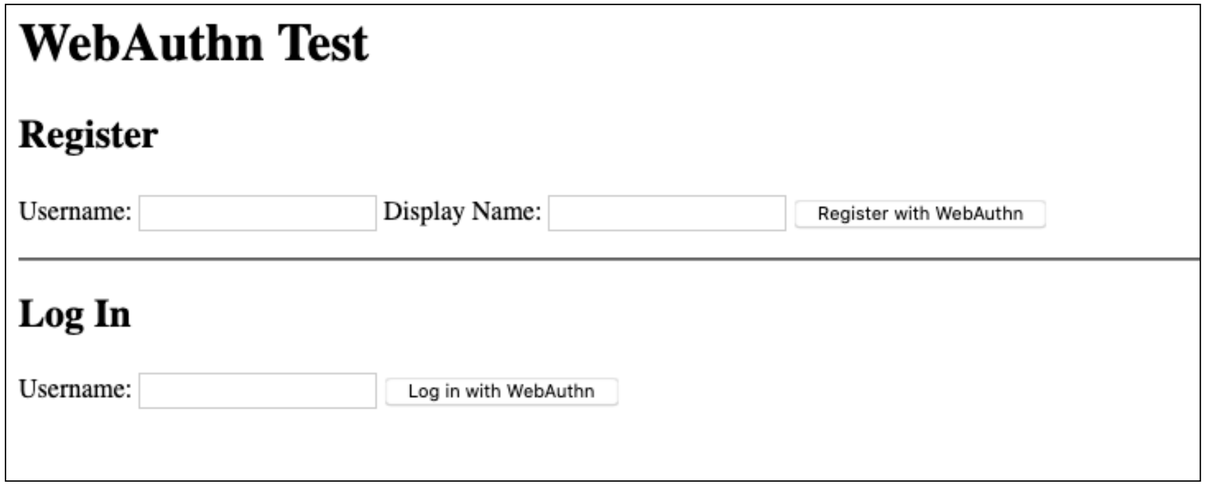
[2] Yubico, “Fido2/webauthn overview.” https://developers.yubico.com/WebAuthn/ WebAuthn Developer Guide/Overview.html, 2020

Execution of Work

Python Test Server



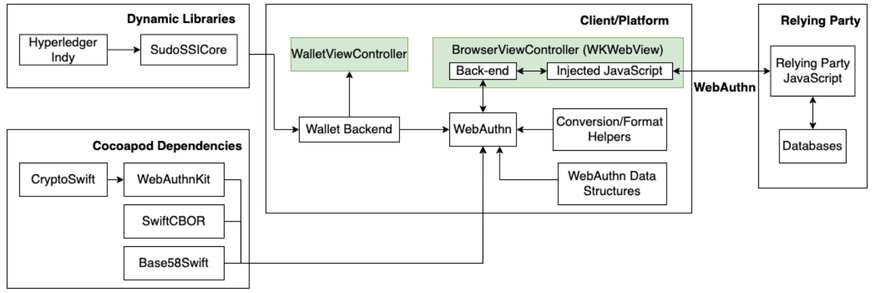
Web UI



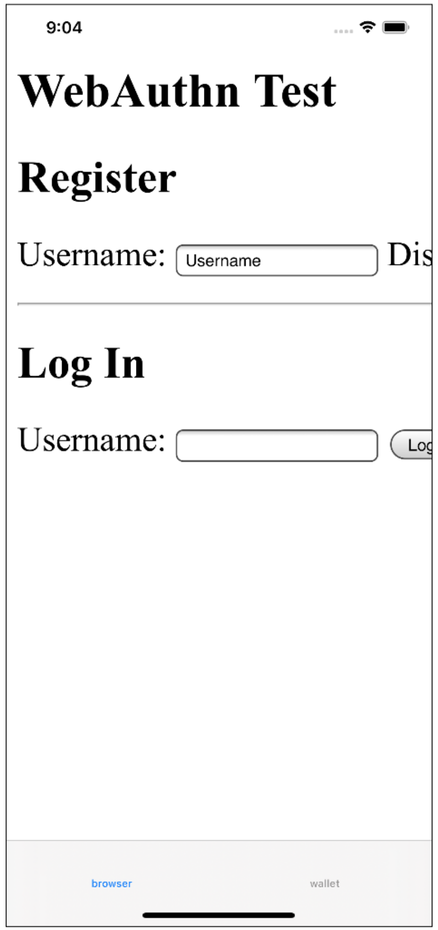
Console

```
nkhor@Nicole-Khors-MacBook-Pro:~/Documents/webauthnserver/py_webauthn/flask_demo$ flask run
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [07/Nov/2020 20:47:02] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [07/Nov/2020 20:47:03] "GET /static/js/lib/base64.js HTTP/1.1" 304 -
127.0.0.1 - - [07/Nov/2020 20:47:03] "GET /static/js/webauthn.js HTTP/1.1" 304 -
127.0.0.1 - - [07/Nov/2020 20:47:03] "GET /static/css/base.css HTTP/1.1" 304 -
127.0.0.1 - - [07/Nov/2020 20:47:48] "POST /webauthn_begin_activate HTTP/1.1" 200 -
```

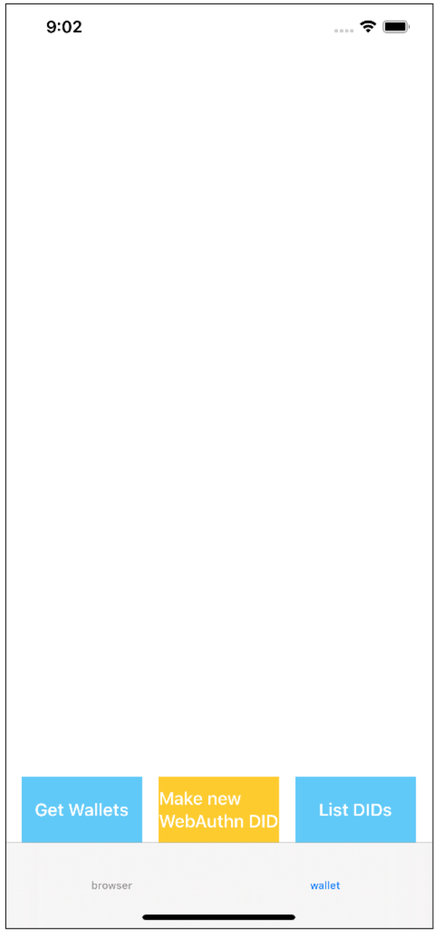
Client Application



Tab View

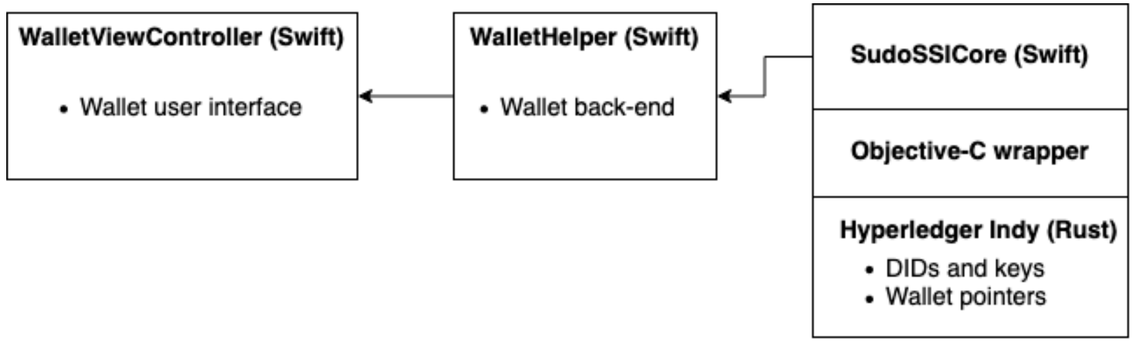


Tab 1

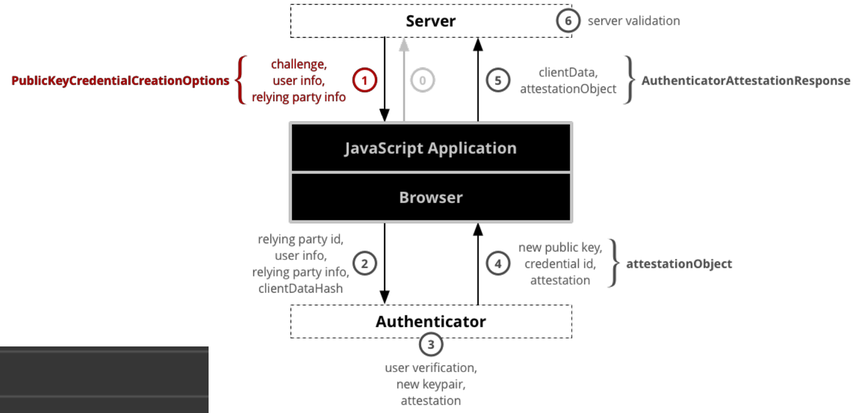


Tab 2

Wallet Modules



Receiving the Credential Options

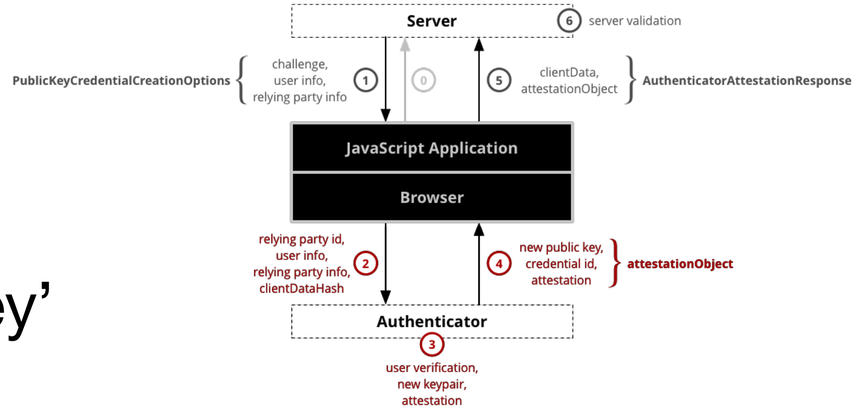



```
Called navigator.credentials.create()
▼ Object
  ▼ O publicKey: Object
      S attestation: "direct"
    ▶ O challenge: Uint8Array [97, 229, 120, 174, 248, 203, 241, 33, 102, 165, …] (32)
    ▶ O excludeCredentials: [] (0)
    ▶ O extensions: {webauthn.loc: true}
    ▶ O metadata: {origin: "http://127.0.0.1:5000"}
    ▶ O pubKeyCredParams: [{alg: -7, type: "public-key"}, {alg: -257, type: "public-key"}, {alg: -37, type: "public-key"}] (3)
    ▶ O rp: {id: "localhost", name: "webauthn demo localhost"}
      N timeout: 60000
    ▶ O user: {displayName: "Display", icon: "https://example.com", id: Uint8Array, name: "Username"}
    ▶ Object Prototype
  ▶ Object Prototype
```

Web Inspector console

Building the Attestation Response

Creating the signature



- No signature method in Hyperledger Indy
- Signatures created with Apple's key system - 'SecKey'
- Required a representation of the private key

```
Wallet created
Found cached wallet 'my-wallet' with handle '3'
Listed DIDs with this response:
[
    {"did":"WPyYSq9j1tYcTZb9DHMUNx","verkey":"H2GcmPLQ1yKt9Rjq5dvZLkQ1Qjq8gsDvjWdr3Vqt4jNi"
    ,"privkey":"2m3ox8LgCKGvmcW7n68ucsxeiS1ue3PDfdZKcniWoybihgYBvxxz3Y6ArMoHzvXAkXnqc811rT8wppDuk9wdhdjg"
    ,"tempVerkey":null,"metadata":"eyJMQUJFTCI6ImZpcnN0X2RpZCJ9"},
    {"did":"CzQjjAjvNbcp9qRBrtyzCT","verkey":"7Y2FvvF7ggLA3QDxixqJ1HZWhUACCkAsfLvvmAvDL14L"
    "privkey":"5AftBdu7wpnoPNW6jdhVaZ7rzNvZi5wSmU6ir3T1jT28fkZk37mnWSz9omvHHXhCRk6iqqxVcfcJvGwyTjx1SGkx"
    ,"tempVerkey":null,"metadata":"eyJMQUJFTCI6ImZpcnN0X2RpZCJ9"},
```

Output from listing DIDs

Building the Attestation Response

Creating the signature

- ‘SecKeyCreateWithData’ created a ‘SecKey’ from an external representation
- ‘bufferAsData’ must be in ANSI X9.63 format, or 04 || X || Y || K

```
guard let privKey = SecKeyCreateWithData(bufferAsData as NSData, [
    kSecAttrKeyType: kSecAttrKeyTypeEC,
    kSecAttrKeyClass: kSecAttrKeyClassPrivate,
    kSecAttrKeySizeInBits: 256
    ] as NSDictionary, &error) else {
        throw error!.takeRetainedValue() as Error
}
```

Routine to create a ‘SecKey’ from an existing key

Guessing X, Y and K for ANSI X9.63

- Elliptic Curve Cryptography:
  - Private key = *k* value with which the generator point, G, is multiplied
  - Public key = (X, Y) coordinate in the subgroup of G = k × G

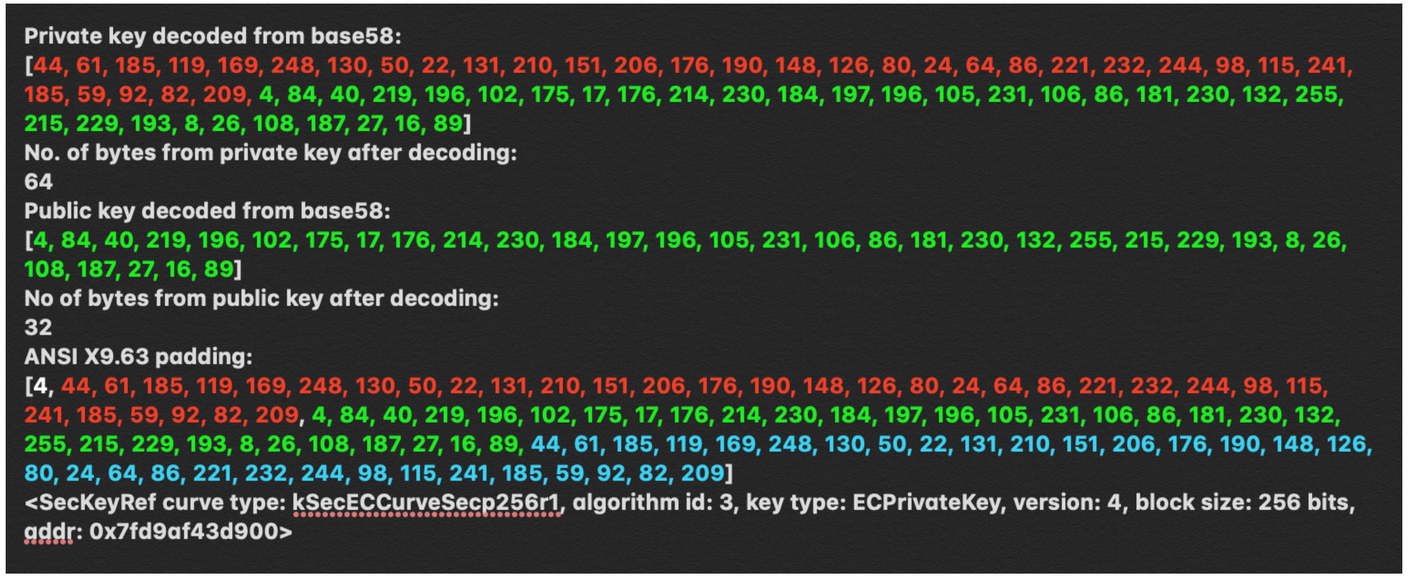
- X, Y = entire public key
- K = first 32 bytes of the private key

```
Private key decoded from base58:
[44, 61, 185, 119, 169, 248, 130, 50, 22, 131, 210, 151, 206, 176, 190, 148, 126, 80, 24, 64, 86, 221, 232, 244, 98, 115, 241,
185, 59, 92, 82, 209, 4, 84, 40, 219, 196, 102, 175, 17, 176, 214, 230, 184, 197, 196, 105, 231, 106, 86, 181, 230, 132, 255,
215, 229, 193, 8, 26, 108, 187, 27, 16, 89]
No. of bytes from private key after decoding:
64
Public key decoded from base58:
[4, 84, 40, 219, 196, 102, 175, 17, 176, 214, 230, 184, 197, 196, 105, 231, 106, 86, 181, 230, 132, 255, 215, 229, 193, 8, 26,
108, 187, 27, 16, 89]
No of bytes from public key after decoding:
32
ANSI X9.63 padding:
[4, 4, 84, 40, 219, 196, 102, 175, 17, 176, 214, 230, 184, 197, 196, 105, 231, 106, 86, 181, 230, 132, 255, 215, 229, 193, 8,
26, 108, 187, 27, 16, 89, 44, 61, 185, 119, 169, 248, 130, 50, 22, 131, 210, 151, 206, 176, 190, 148, 126, 80, 24, 64, 86, 221,
232, 244, 98, 115, 241, 185, 59, 92, 82, 209]
Error Domain=NSOSStatusErrorDomain Code=-50 "EC private key creation from data failed"
UserInfo={NSDescription=EC private key creation from data failed}
Signing failed.
```

Failed ‘SecKey’

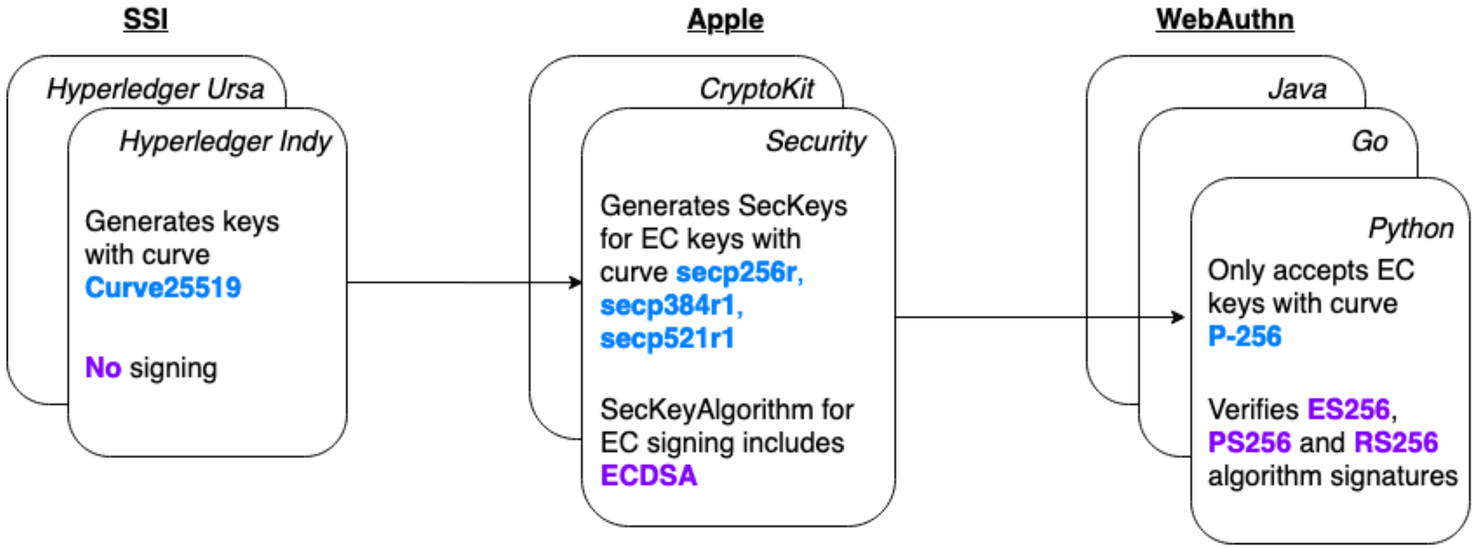
Successful SecKey

- X, Y = entire **private** key
- K = first 32 bytes of the private key **again**



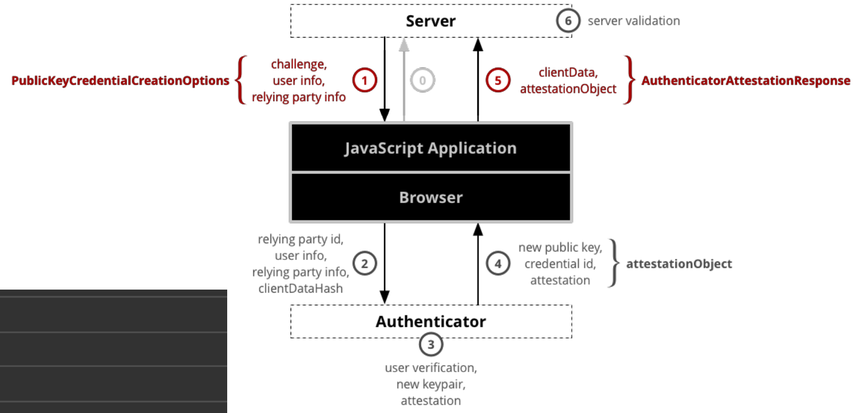
Successful ‘SecKey’

Cryptographic Incompatibility



Cryptography systems between modules

Sending the Attestation Response




```
Called navigator.credentials.create()
▶ {publicKey: Object}
Registration response received.
▶ {rawId: Array, id: "__5ZADEAQQBmAG4AUwBpADcAZQBiAHAAUwBQAEcAagBZAGsAcABFADgATgA", response: Object, type: "public-key"}
▶ [73, 150, 13, 229, 136, 14, 140, 104, 116, 52, …] (126)
▶ Uint8Array [73, 150, 13, 229, 136, 14, 140, 104, 116, 52, …] (126)
Now checking for registration response.
▶ {rawId: Array, id: "__5ZADEAQQBmAG4AUwBpADcAZQBiAHAAUwBQAEcAagBZAGsAcABFADgATgA", response: Object, type: "public-key", getClientExtensionResults: function, …}
Received new credential from browser!
▼ Object
  ▶ f getClientExtensionResults: function()
    S id: "__5ZADEAQQBmAG4AUwBpADcAZQBiAHAAUwBQAEcAagBZAGsAcABFADgATgA"
  ▶ f isUserVerifyingPlatformAuthenticatorAvailable: function()
  ▶ O rawId: [95, 95, 53, 90, 65, 68, 69, 65, 81, 81, …] (59)
  ▼ O response: Object
    ▼ O attestationObject: ArrayBuffer
        N byteLength: 291
      ▶ ArrayBuffer Prototype
    ▶ O clientDataJSON: [123, 34, 116, 121, 112, 101, 34, 58, 34, 119, …] (119)
    ▶ Object Prototype
    S type: "public-key"
  ▶ Object Prototype
▶ Server validation of credential failed: – "Registration failed. Error: Registration rejected. Error: Unsupported algorithm.."
```

Web Inspector console

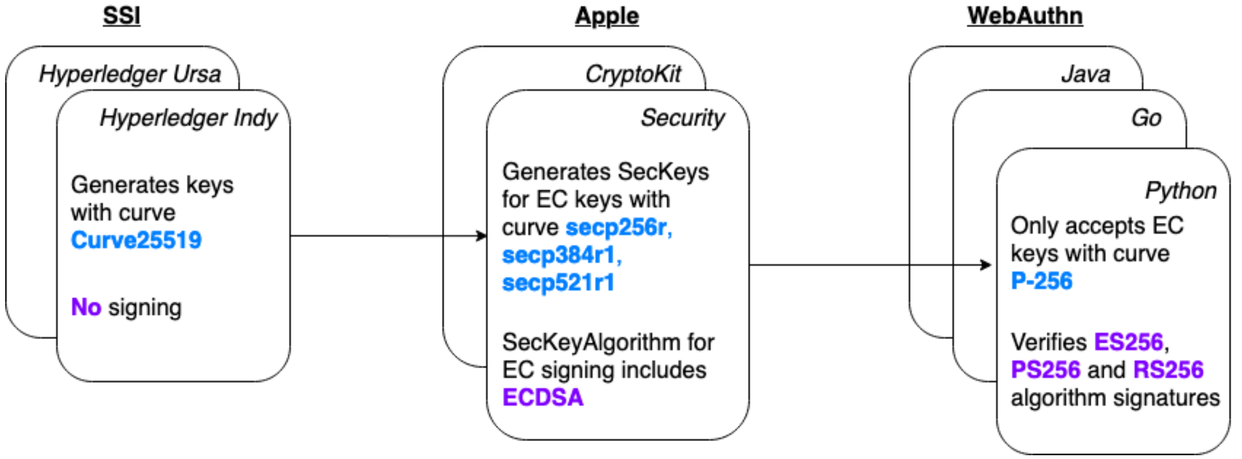
Registration Verification

- Rejects at verifying signature
  - ‘Unsupported algorithm’
- If self-attestation is accepted, no other relevant checks

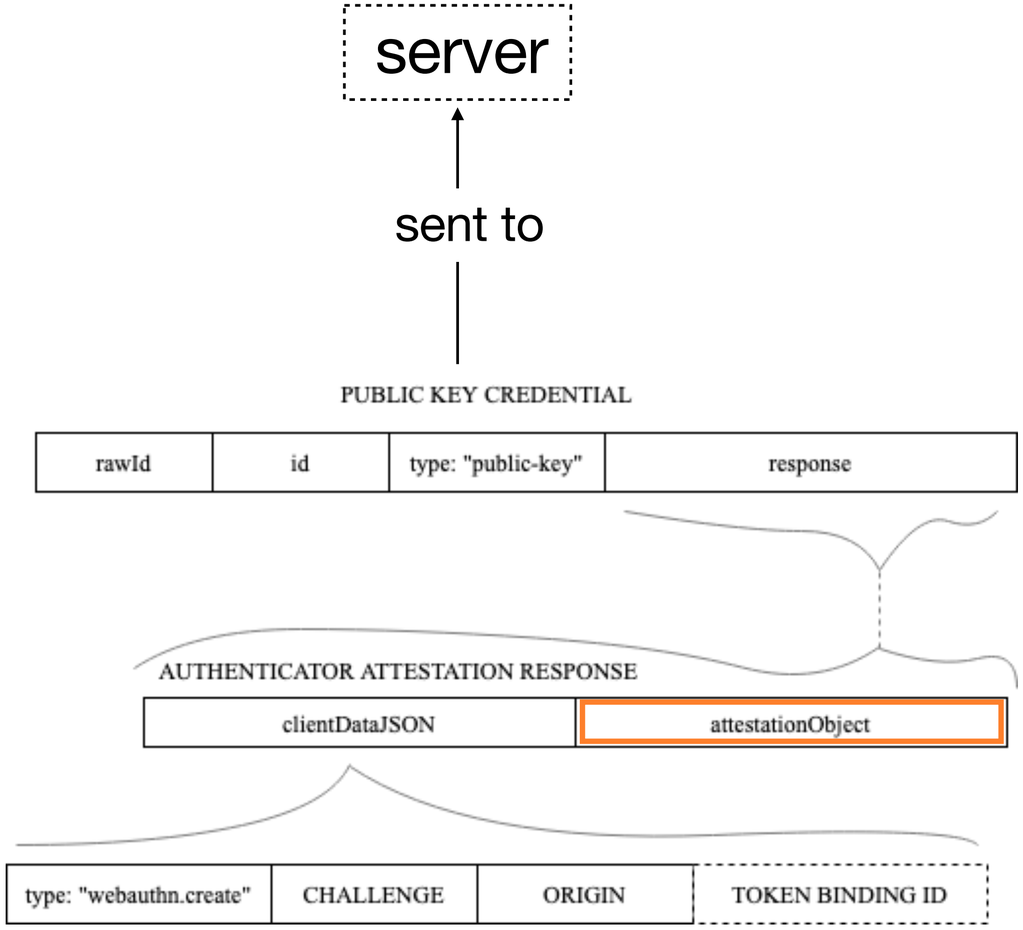
Step	Procedure
1	Let JSONtext = response.clientDataJSON decoded from UTF-8
2	C = parsed JSONtext through JSON parser
3	Verify C.type == ”webauthn.create”
4	Verify C.challenge == challenge originally generated
5	Verify C.origin == origin of relying party
6	Verify C.tokenBinding.status == state of token binding for the TLS connection over which the assertion was obtained
7	Let hash = response.clientDataJSON hash using SHA-256
8	CBOR decode the attestationObject to obtain the attestation statement
9	Verify ipIdHash == SHA-256 hash of relying party identifier
10	Verify the User Present bits in flags == 1
11	Verify User Verified bit in flag == 1 if user verification is required
12	Verify extensions are as expected
13	Determine attestation format from ’fmt’ in attestation object
14	Verify attStmt is the correct attestation statement by verifying the attestation signature and format
15	If validation is successful, obtain a list of acceptable trust anchors
16	Assess the attestation trustworthiness. If self-attestation was used, check if self-attestation is acceptable under the relying party policy.
17	Check credentialId not yet registered to any other user
18	If attStmt verified successfully and is trustworthy, register new credential with user identifier provisioned by server initially.
19	If attestation statement was not successful or not trustworthy, fail registration ceremony.

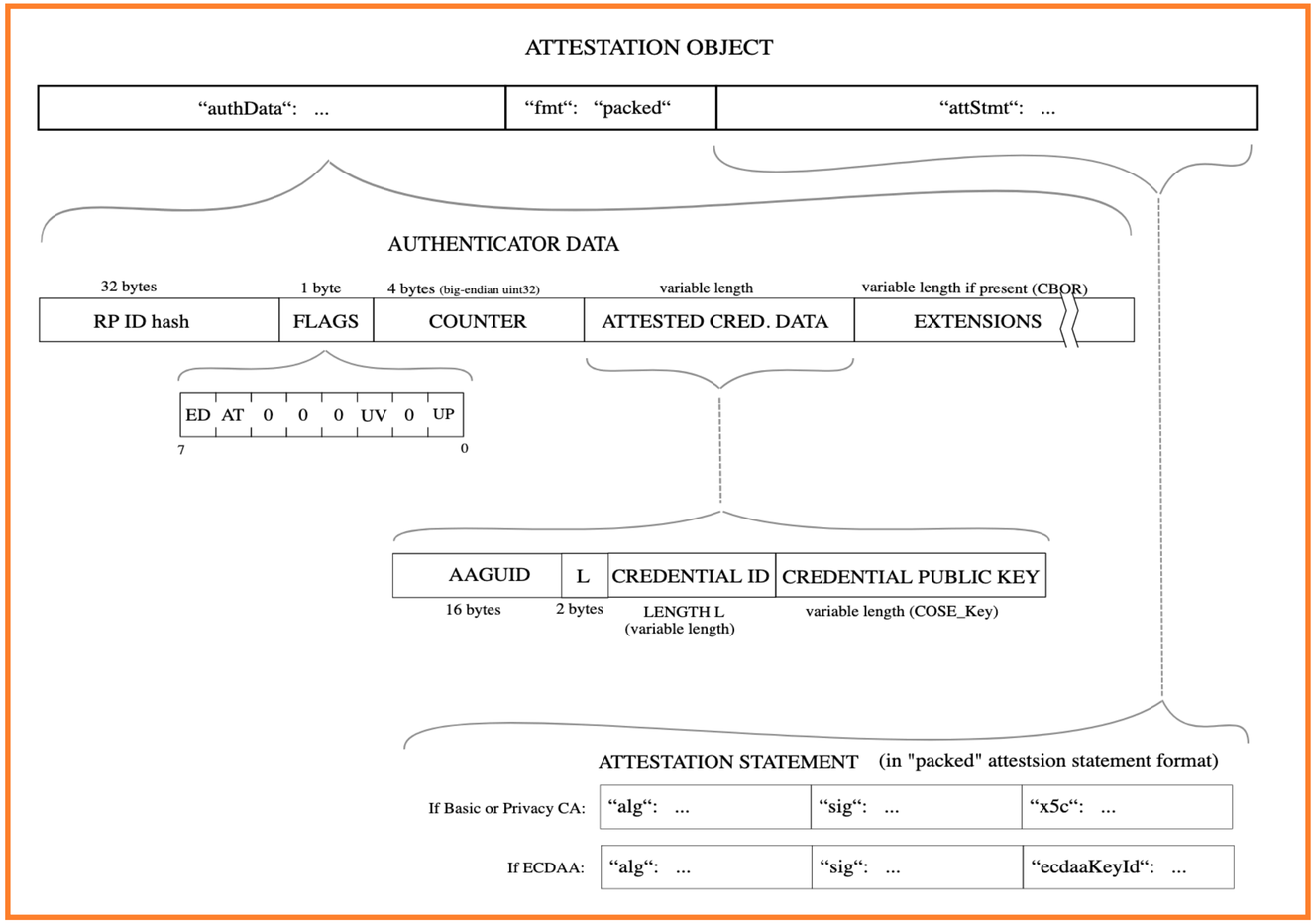
Possible Future Alternatives

- Implement signatures for Ed25519 keys in Hyperledger Indy
  - Avoids exposing private key
  - Avoids Apple intermediary
- Implement a different curve in Hyperledger Indy
- Use Hyperledger Ursa as cryptographic back end
- Consider other Apple and WebAuthn server libraries



Registration Response

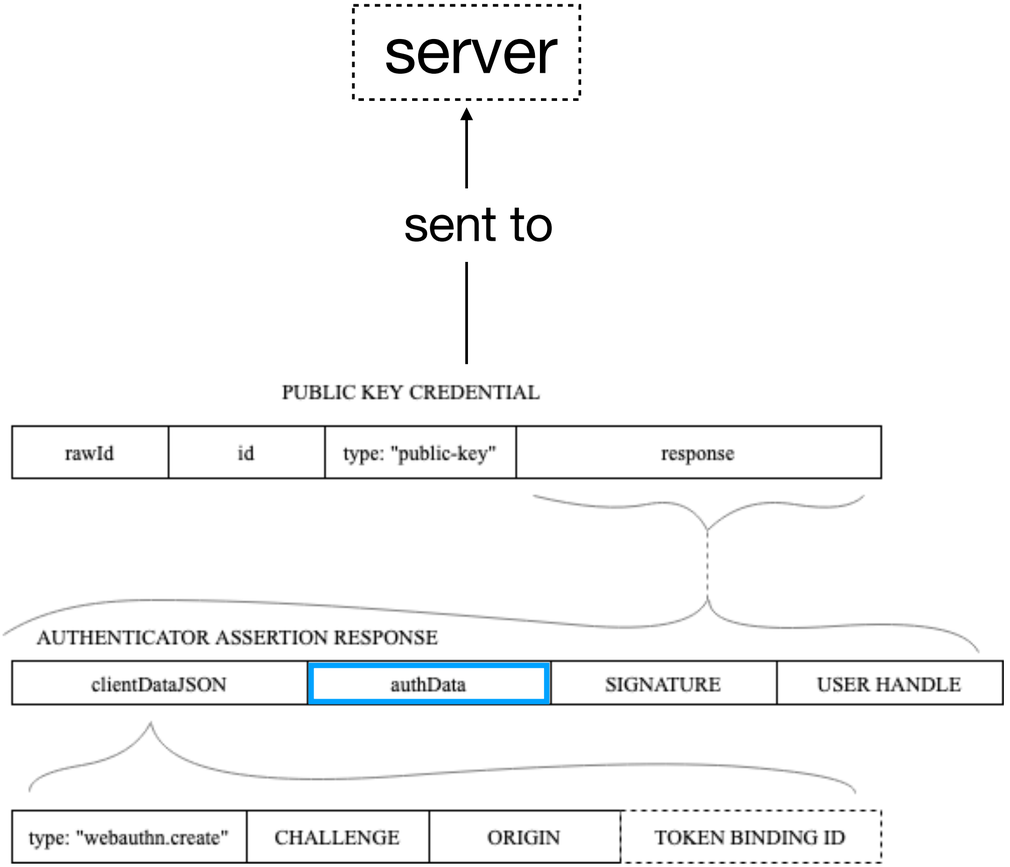


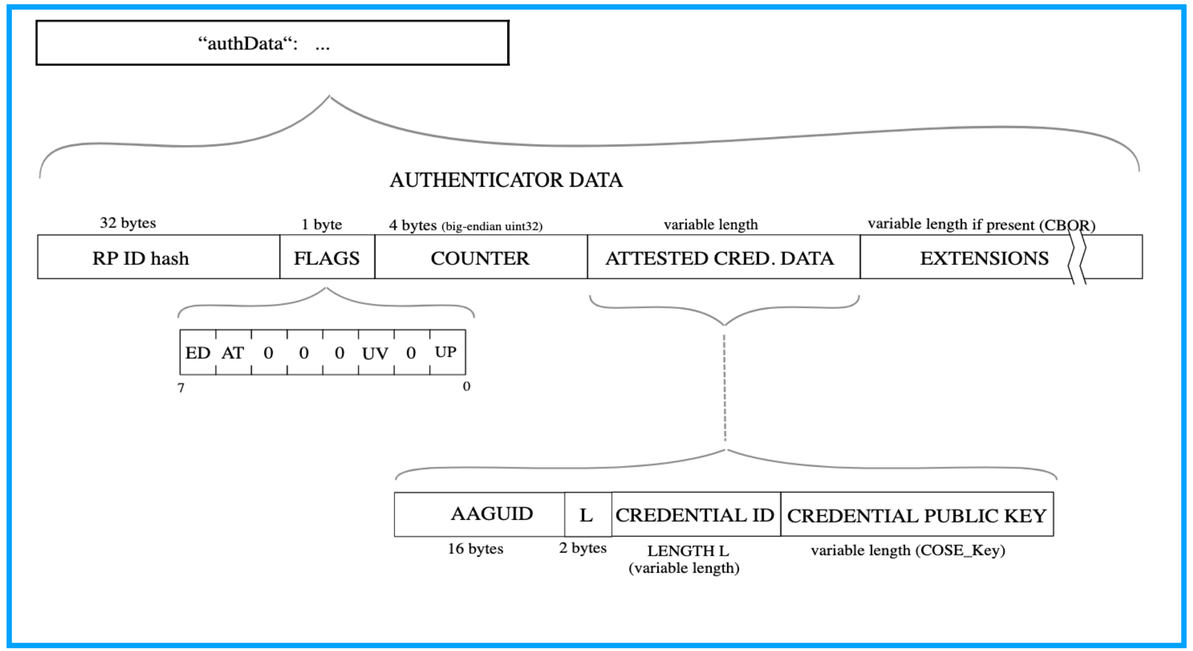


[2]

[2] Yubico, “Fido2/webauthn overview.” https://developers.yubico.com/WebAuthn/ WebAuthn Developer Guide/Overview.html, 2020

Authentication Response

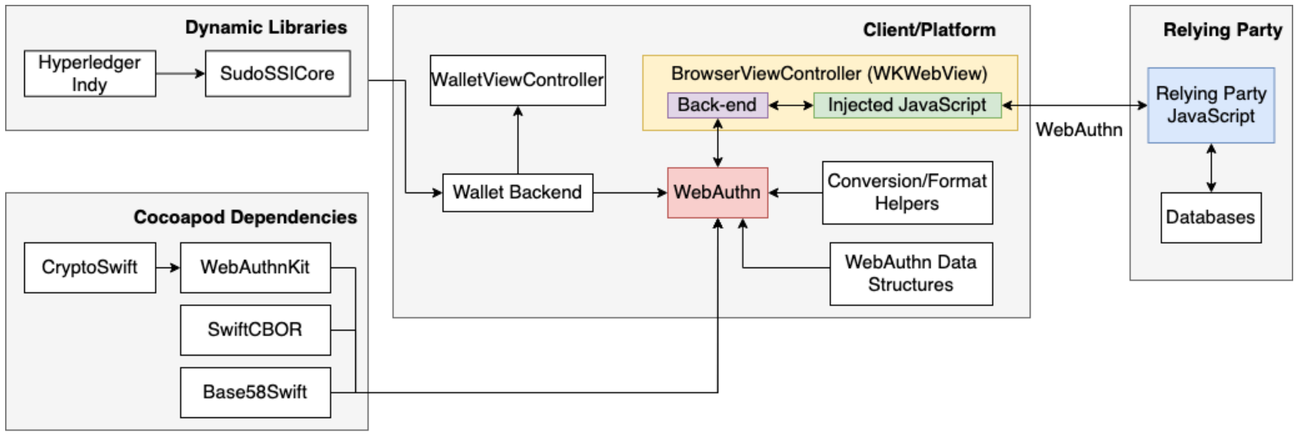


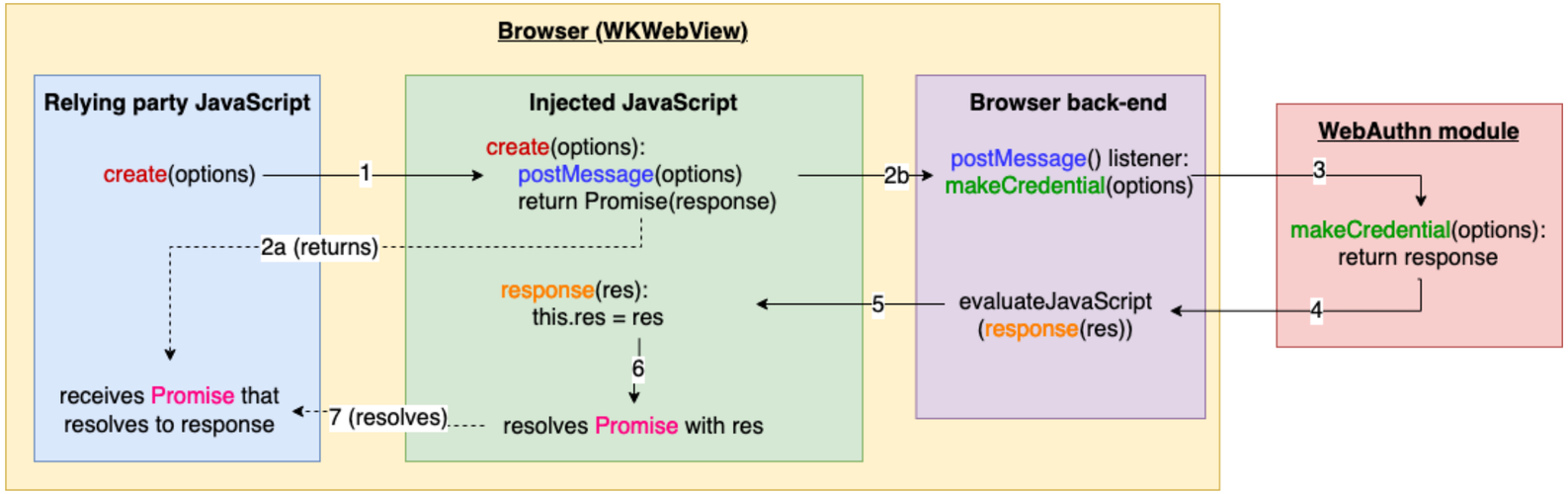


[2]

[2] Yubico, "Fido2/webauthn overview." https://developers.yubico.com/WebAuthn/ WebAuthn Developer Guide/Overview.html, 2020

Client and Server Communication via Script Injection





Interesting SSI Decisions

- Borrowing another device's AAGUID
    - Indy wallet does not have an AAGUID
    - Could be anti-fingerprinting
- WebAuthn DID provisioning
    - One DID for all WebAuthn services vs. one DID per service