FIDO2 Functional Overview

Deep dive into data flows, attestation, and passwordless authentication

Luke Walker, Senior Solutions Architect, Yubico

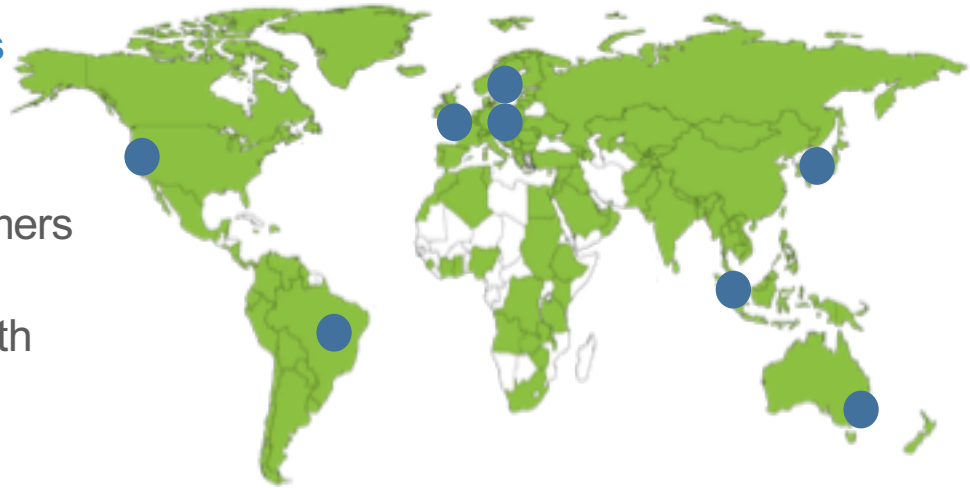
Agenda

- Introduce FIDO Authentication, CTAP, and WebAuthn
- Registration Flow
- What is Attestation?
- Resident Keys
- Second-Factor Authentication Flow
- Passwordless Authentication Flow
- Upgrading from U2F
- Deployment Considerations
- Best Practices

Meet Yubico!

Making Secure Access Ubiquitous

- Founded in 2007
- Employees in eight countries
- Thousands of enterprise customers and millions of users globally
- Co-Inventor of U2F standard with Google
- Co-Inventor of FIDO2 (CTAP) standard with Microsoft
- One of nine editors of W3C WebAuthn specification



Google Microsoft facebook PayPal Dropbox salesforce GitHub redhat. LINUX

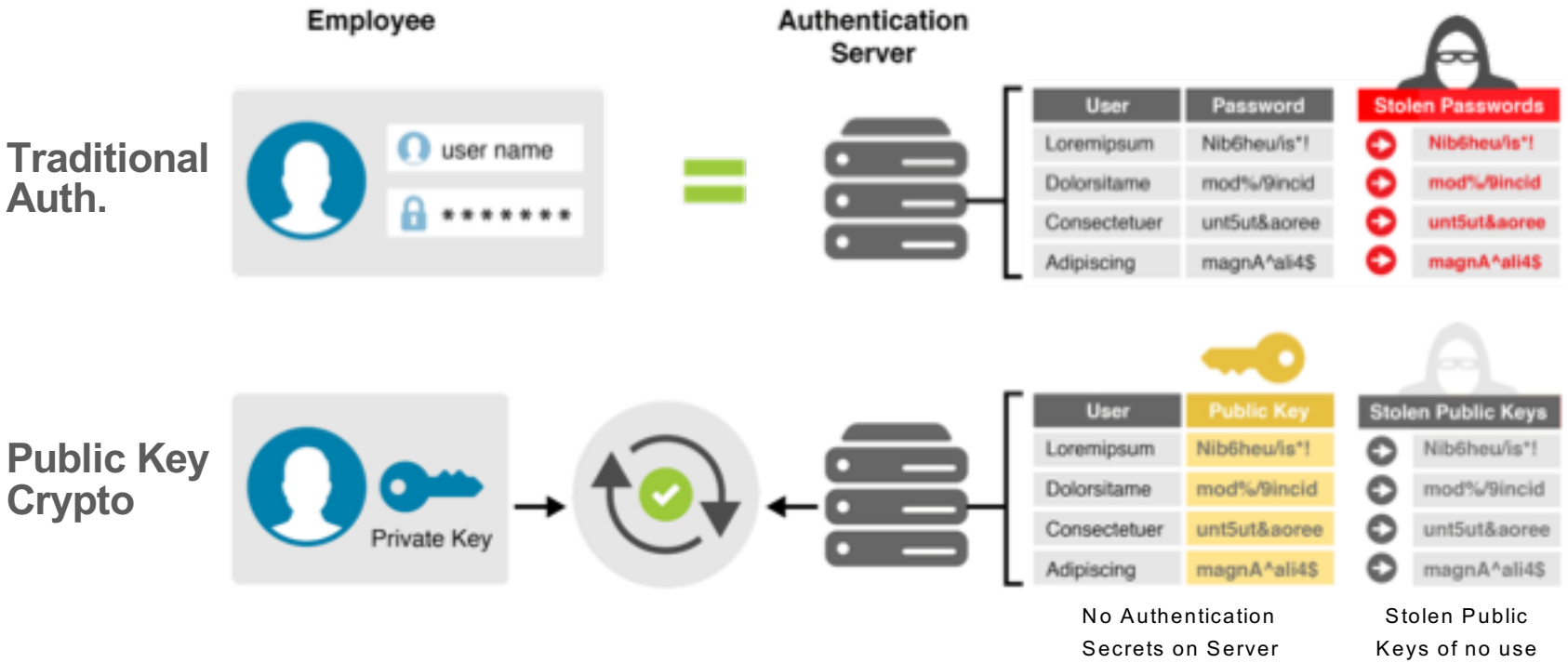
yubico

FIDO2 Authentication

Solving the Password Problem

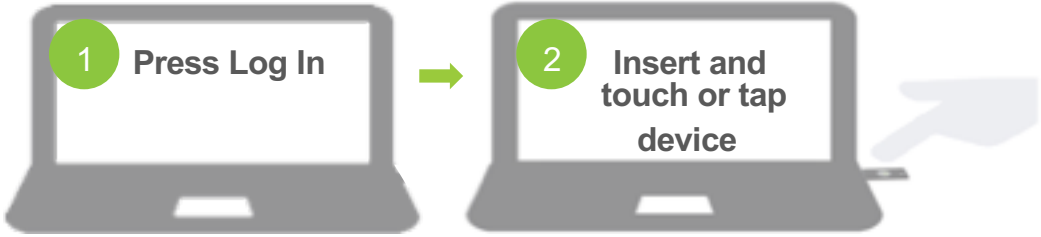
Public Key Crypto = No Secret on Server

Only Public Keys are stored on the server, not private ones



Building a FIDO2 Solution

Simple



Secure

Public Key Crypto

Origin bound keys

User presence

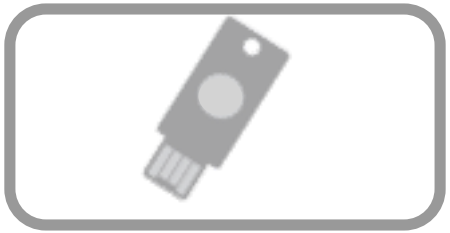
Scalable

Native platform support

Many services,
no shared secrets

FIDO2 Overview

New open authentication standard offering new authentication choices



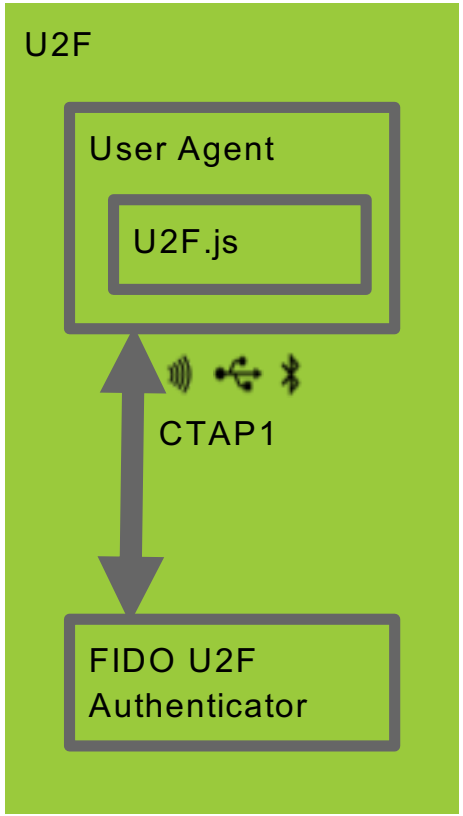
Single Factor: Passwordless
Replaces weak passwords with strong authentication for single factor authentication

Two Factor: Password + Authenticator
Second factor in a two factor authentication solution

Multi-Factor: Passwordless + PIN or Biometric
Combination of a hardware authenticator with touch and a PIN for high assurance such as financial transactions or regulations

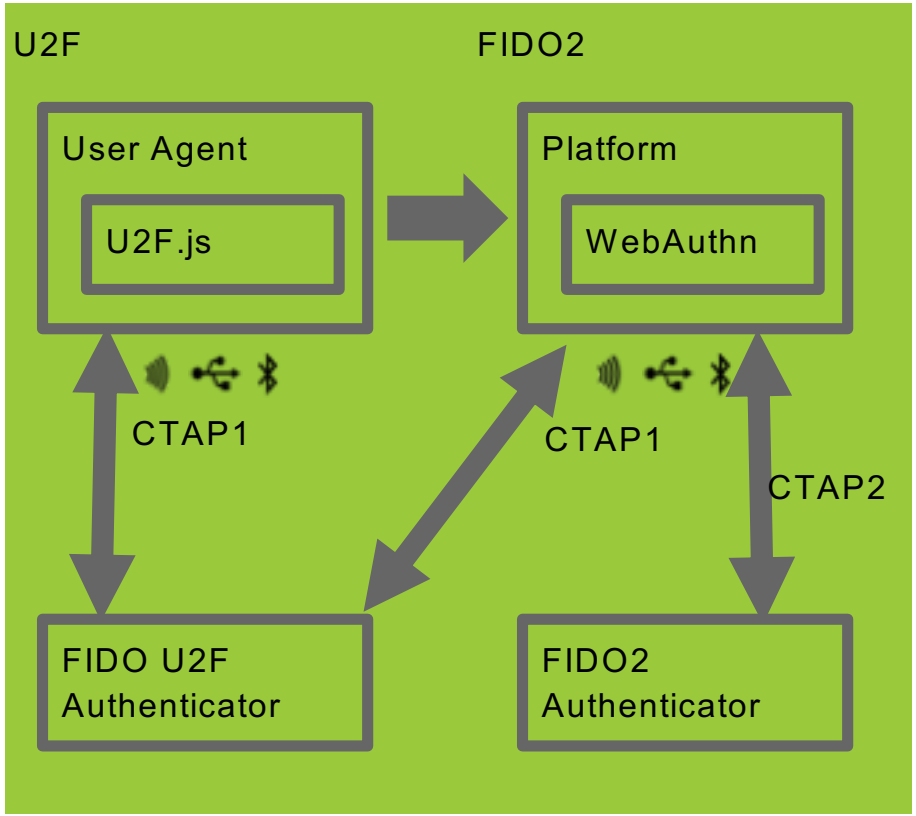
yubico

Why We Needed More than FIDO U2F



- Still requires a username and password
- Doesn’t support user verification
- Limited global acceptance of standard APIs
- Offers a subset of Multi-Factor Authentication

Evolution of FIDO Authentication



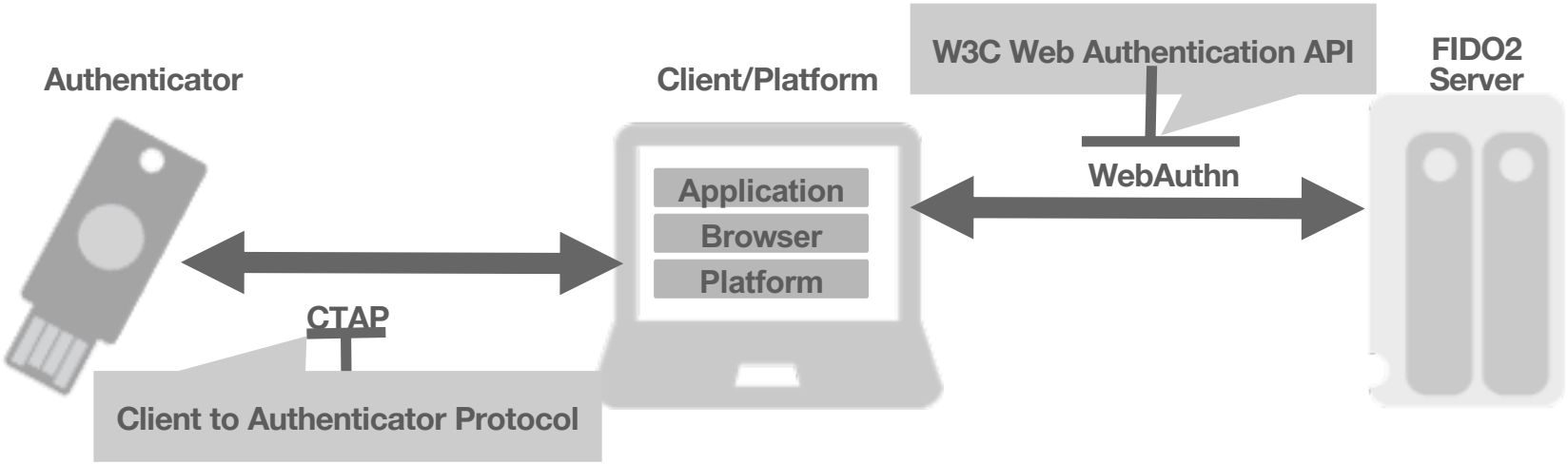
U2F

- **Phishing resistant authentication with user intent**
- **Multi-Factor Authentication (MFA) Subset**
  - Authenticator - something you have
  - Password - something you know

FIDO2

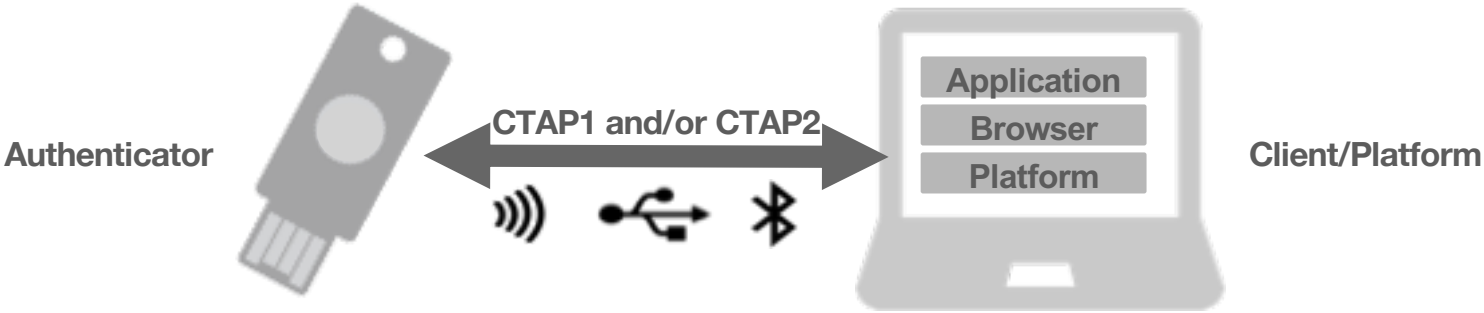
- **True MFA**
  - Authenticator - something you have
  - User verification - something you know (PIN) or are (Biometrics)

How FIDO2 Authentication Works



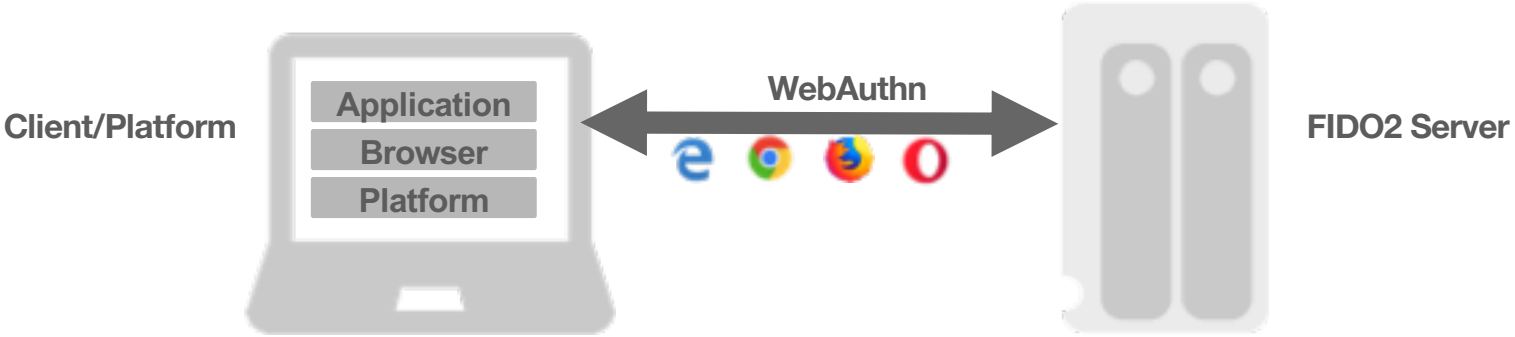
- A set of open standards utilizing public-key cryptography to enable strong first factor, second and multi factor authentication
- FIDO2 = CTAP + WebAuthn

What is CTAP?



- Authenticator generates and securely stores credentials
- Private keys, PINs, and biometric information never leave the authenticator

What is WebAuthn?



- Creation and use of strong public key-based credentials by web applications
- Strongly authenticate users
- All major browsers are on track to implement full Web Authentication APIs

How Does it Work?

Extends W3C Credential Management API and adds a new credential type:

- Password, Federated, PublicKeyCredential

Enables users to:

- Create and register a public key credential for a website
- Authenticate to a website by proving possession of a private key

FIDO2 Functional Overview

Registration and authentication walkthrough

Registration Demo

https://demo.yubico.com/webauthn

Try it yourself:

- **Operating Systems:**
  - Windows
  - Linux
  - Mac
- **Browsers:**
  - Chrome
  - Firefox
  - Edge (Insider builds)
- **YubiKey Models:**
  - YubiKey NEO
  - YubiKey 4 Series (YubiKey 4, YubiKey 4 Nano, YubiKey 4C, YubiKey 4C Nano)
  - FIDO U2F Security Key
  - Security Key by Yubico

yubico

yubico

TRY WEBAUTHN

This demo will let you register and log in to an account using Web Authentication, also known as WebAuthn. This requires a WebAuthn authenticator device, as well as a browser with WebAuthn support.

Start by registering a user, then try logging in.

Register | Login | Login without username | Add authenticator | Delete account

Register using WebAuthn

Username

Authenticator nickname

Enable username-less login

Next

Enter a username to initialize the registration process. A password is not required - instead, you will be prompted to register your authenticator in the next step.

Optionally, you can also add a nickname for your authenticator.

Since this is a demo, registered accounts will last only a limited time.

Registration Highlights

Authenticator | **Client** | **Relying Party**

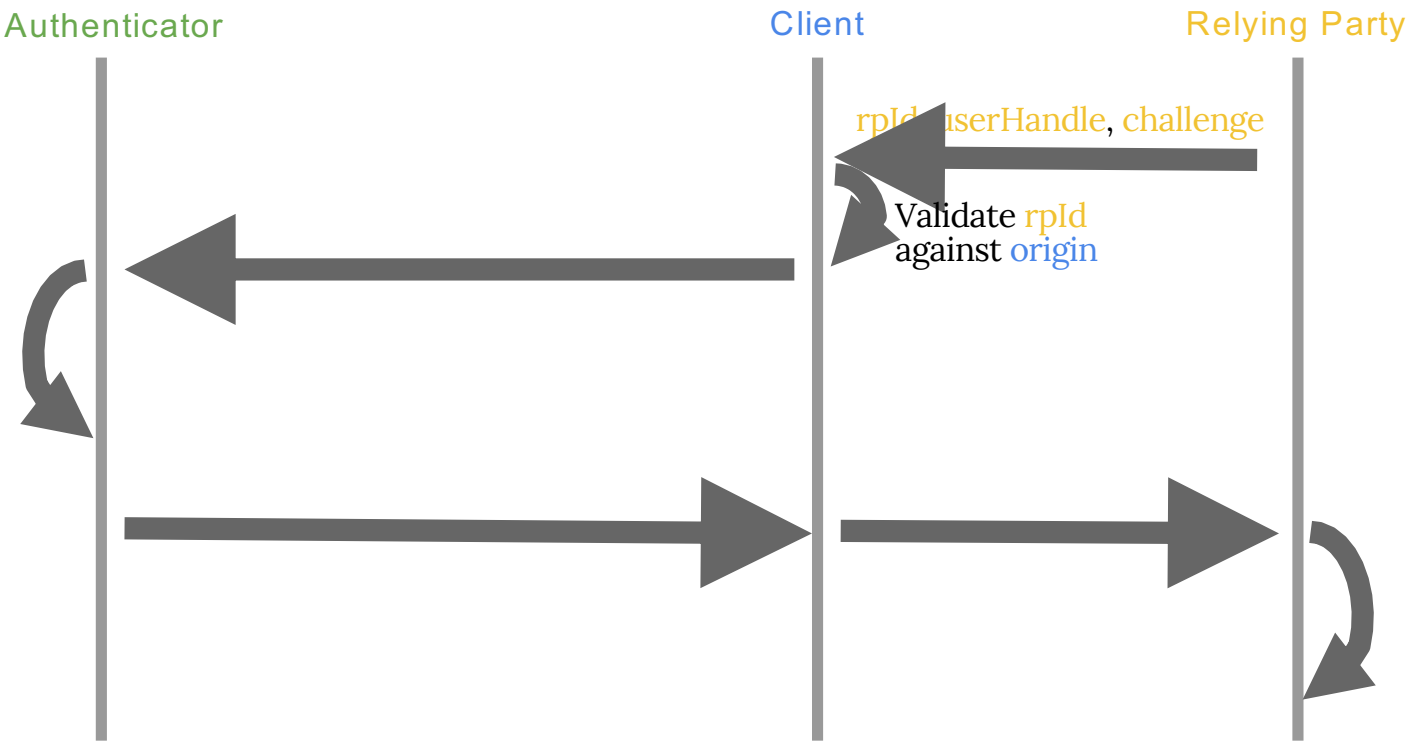
1. Relying Party: Generates a challenge
2. Client: Client validates origin
3. Authenticator: Authenticator checks user presence & consent
4. Authenticator: Creates key pair
5. Relying Party: Verifies attestation signature

Prevents phishing (Relying Party → Client)

Prevents silent tracking (Client → Authenticator)

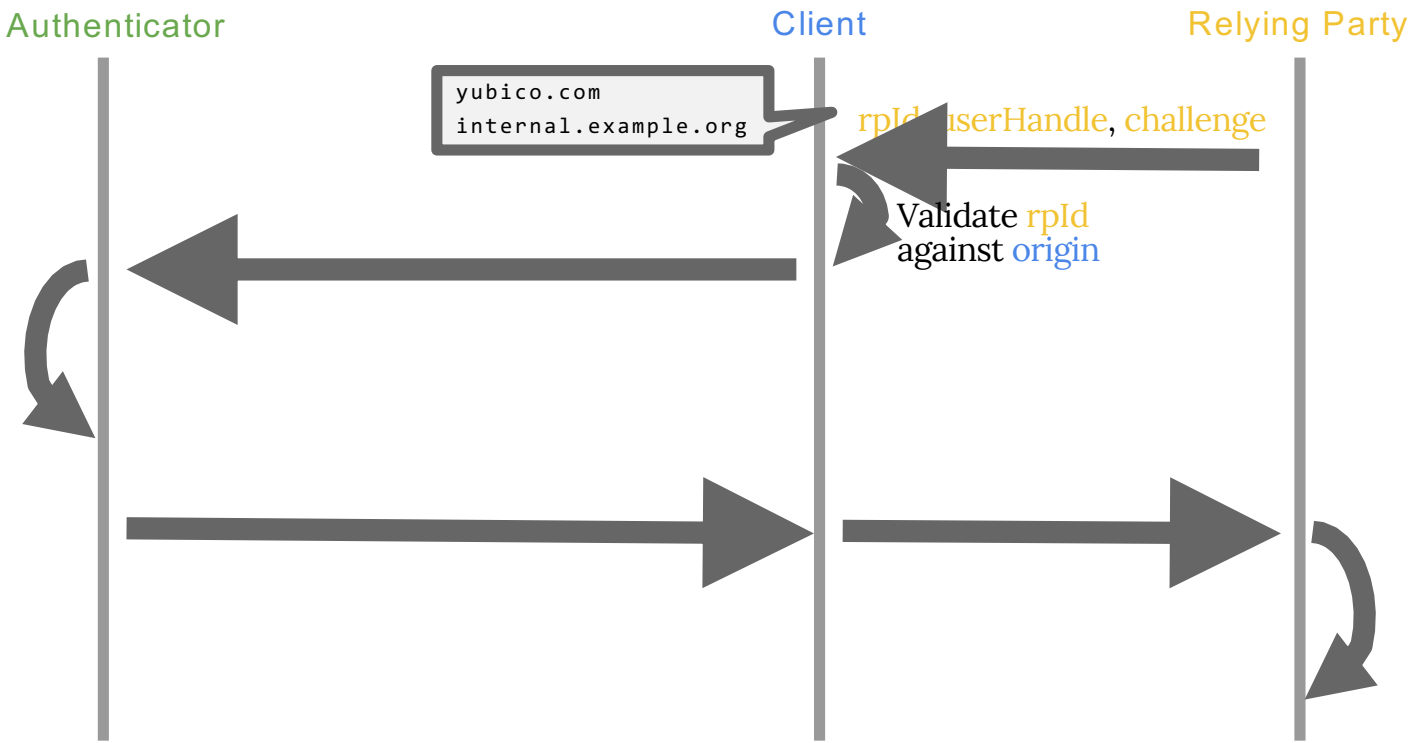
Proof that private key is safe (Authenticator → Relying Party)

Registration Sequence

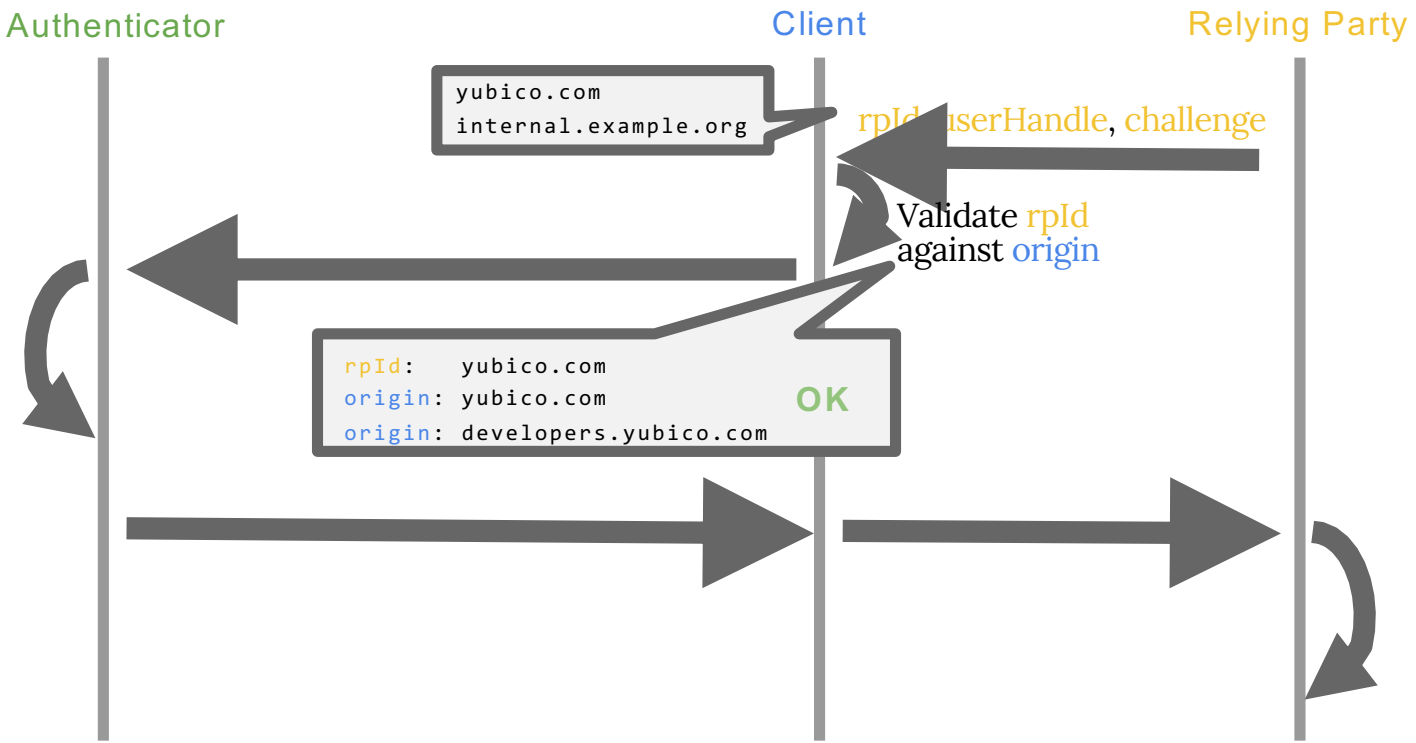


yubico

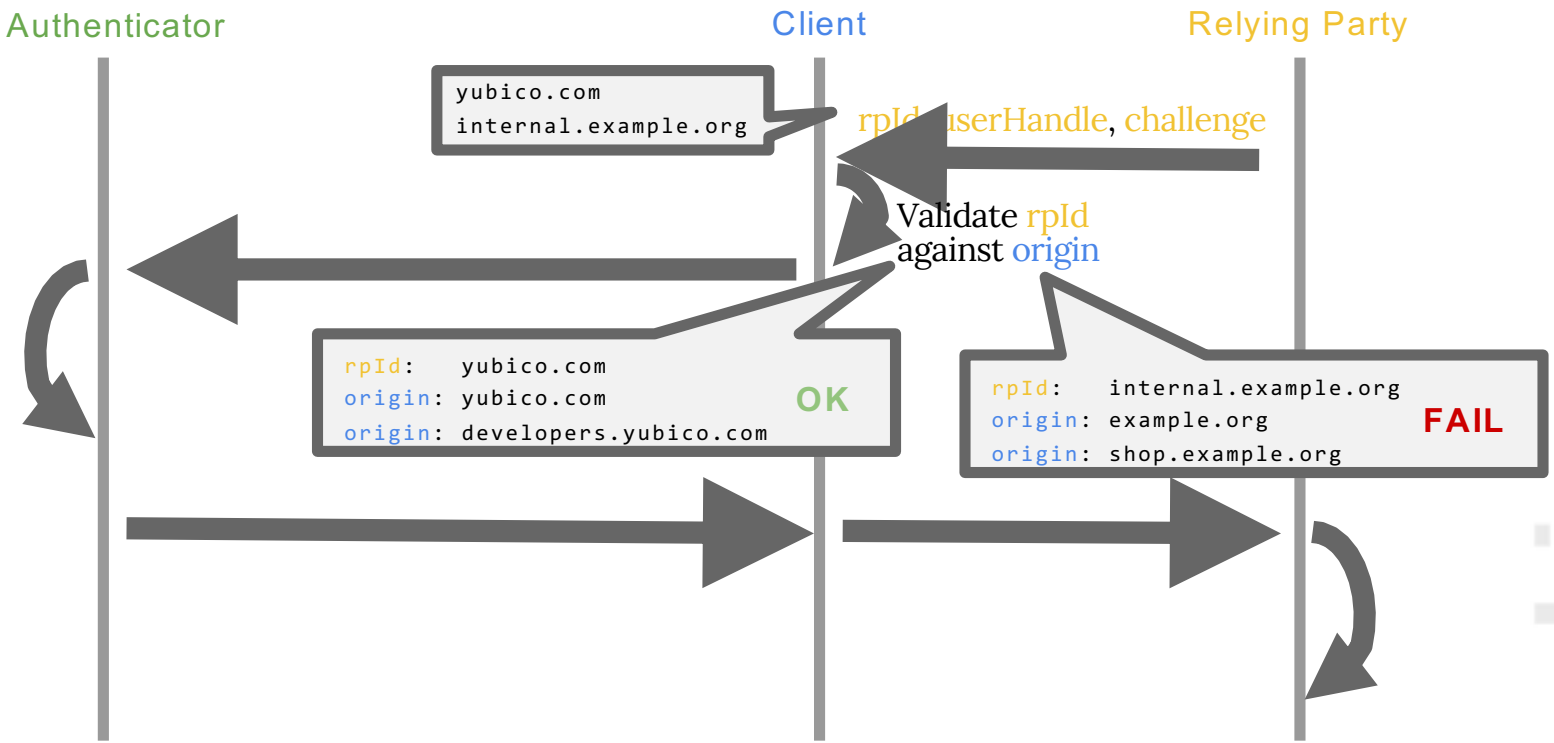
Registration Sequence



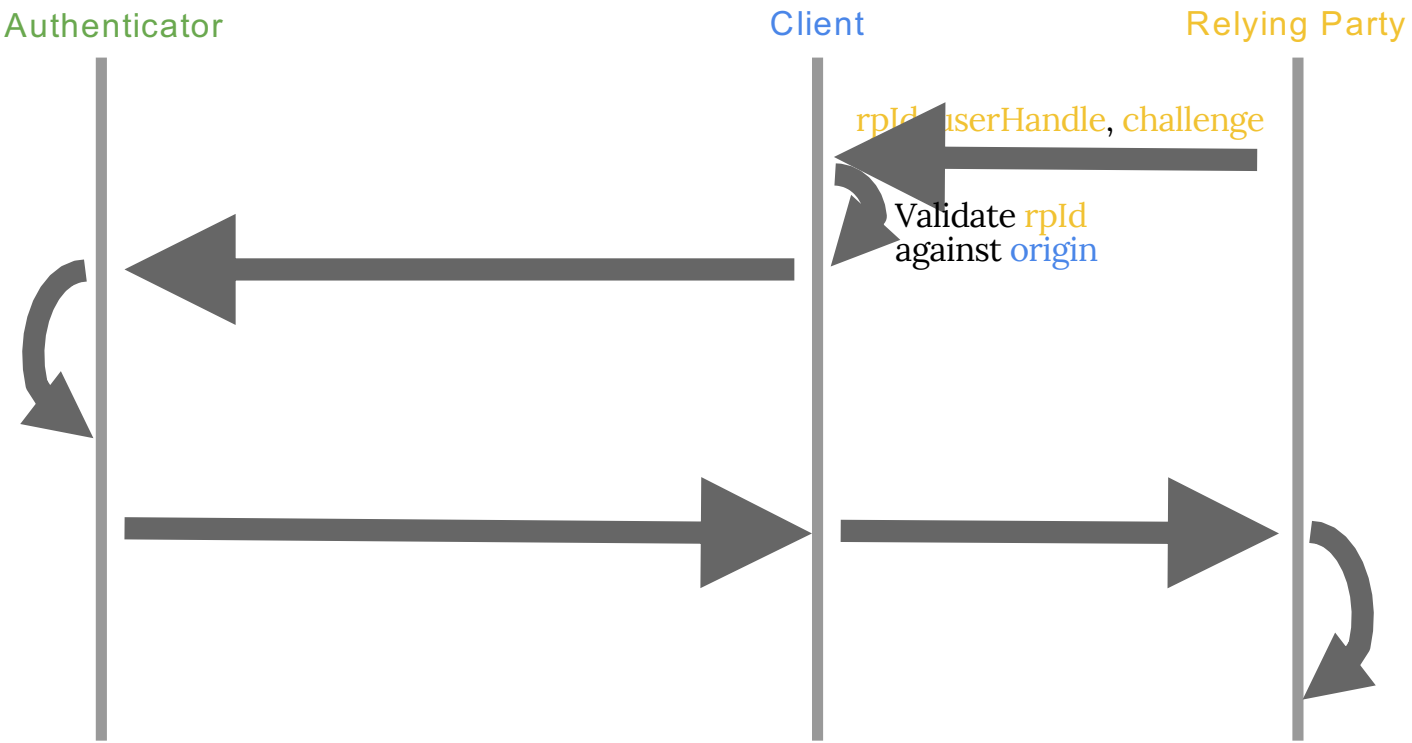
Registration Sequence



Registration Sequence

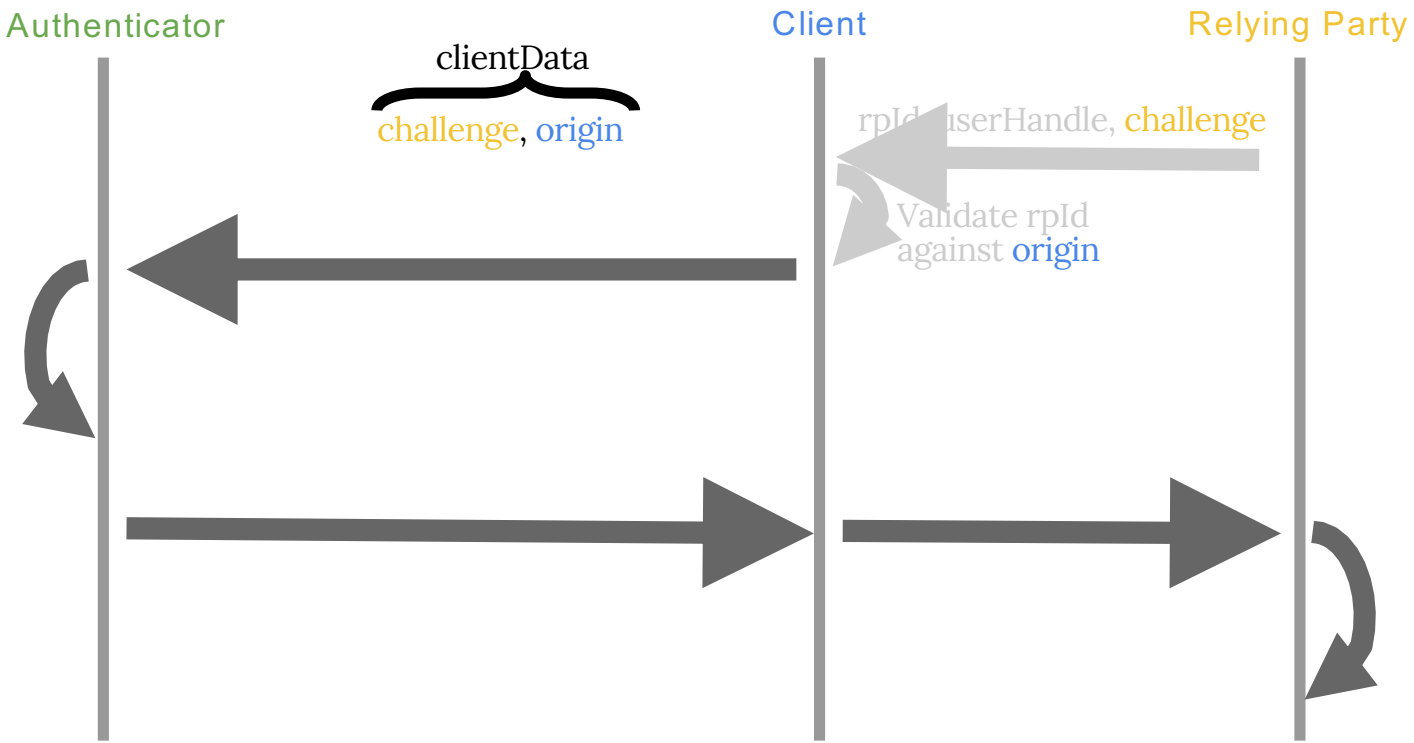


Registration Sequence

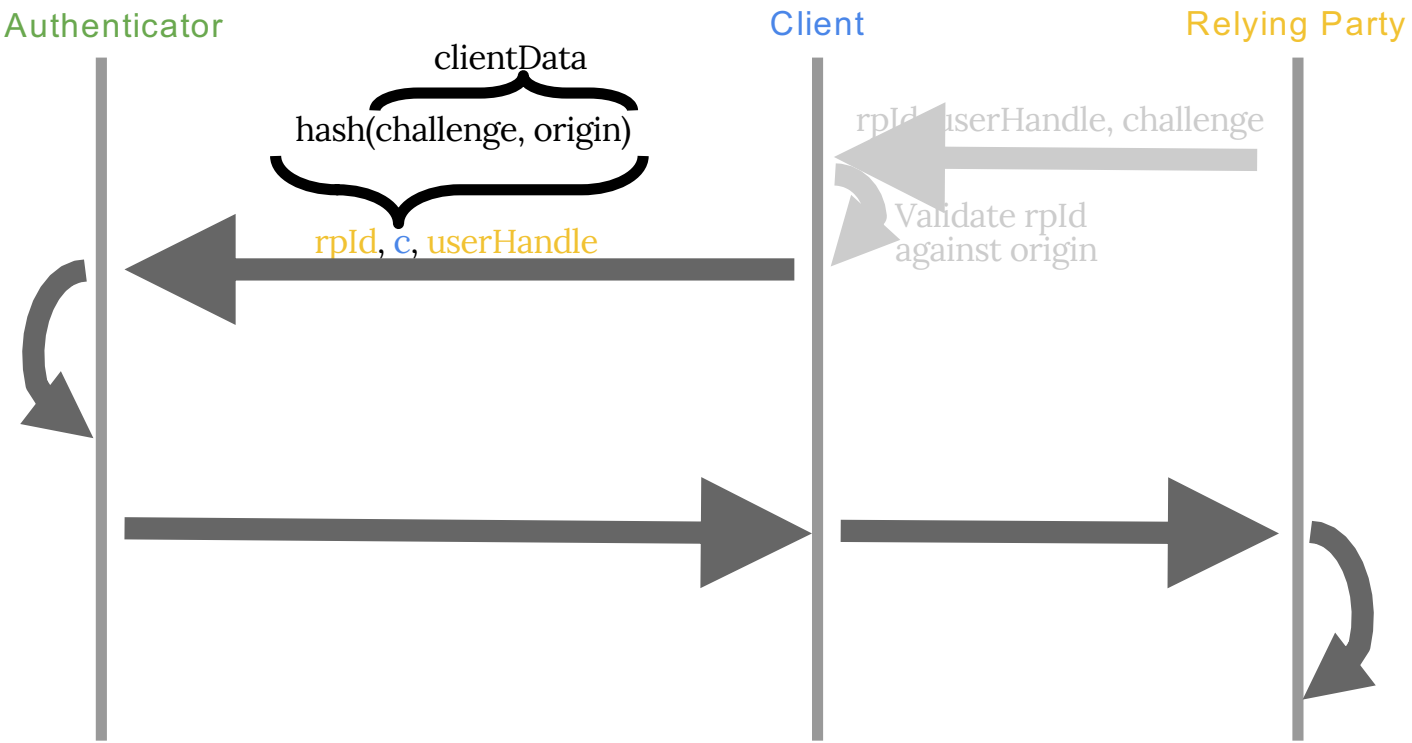


yubico

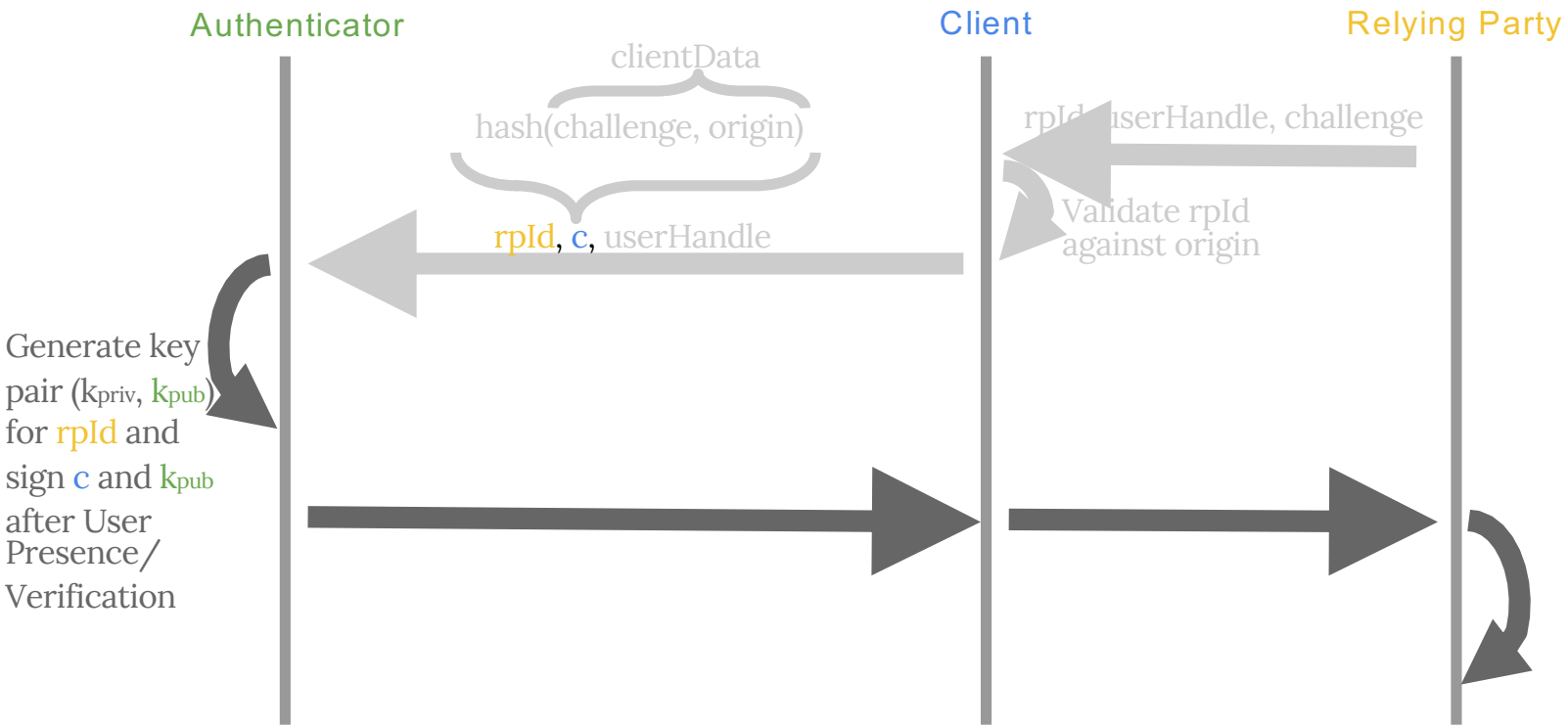
Registration Sequence



Registration Sequence

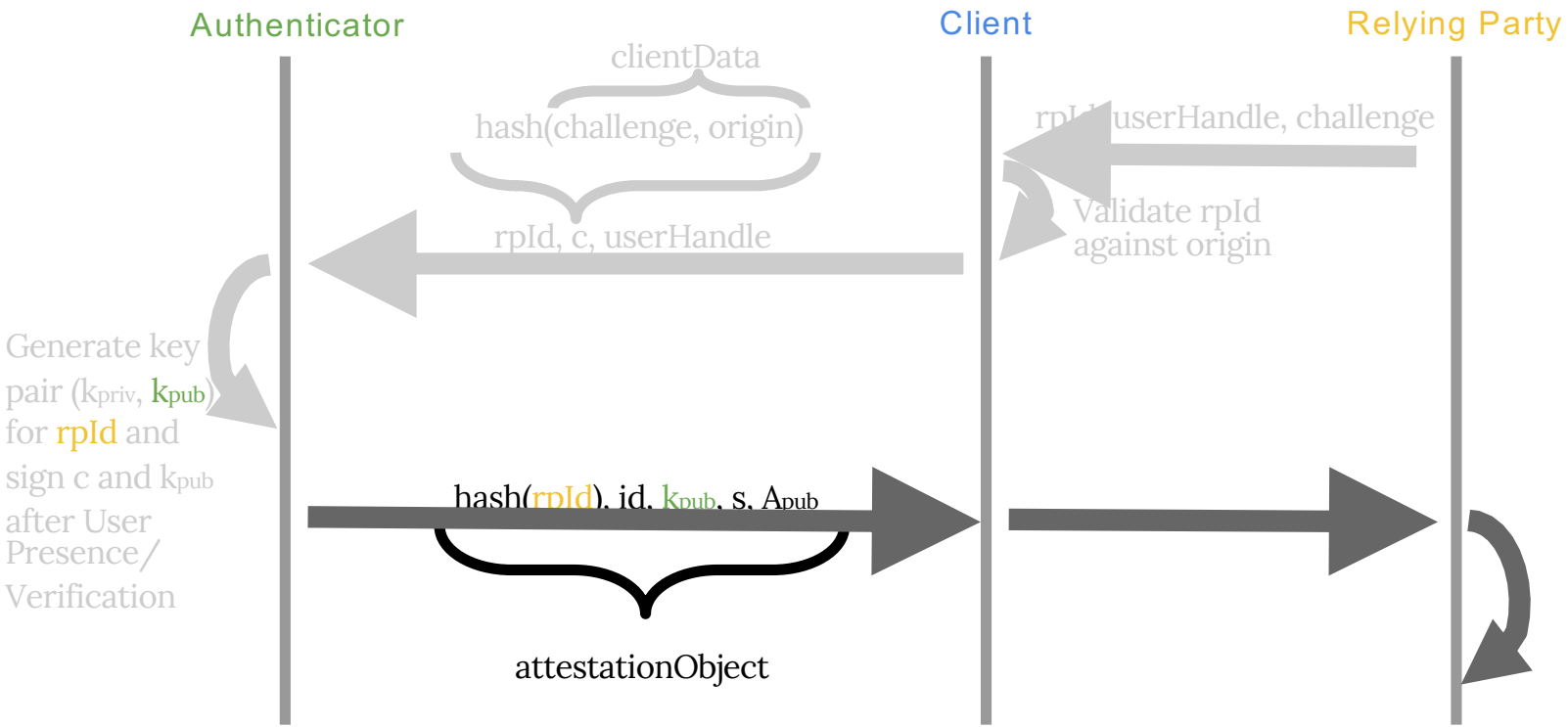


Registration Sequence



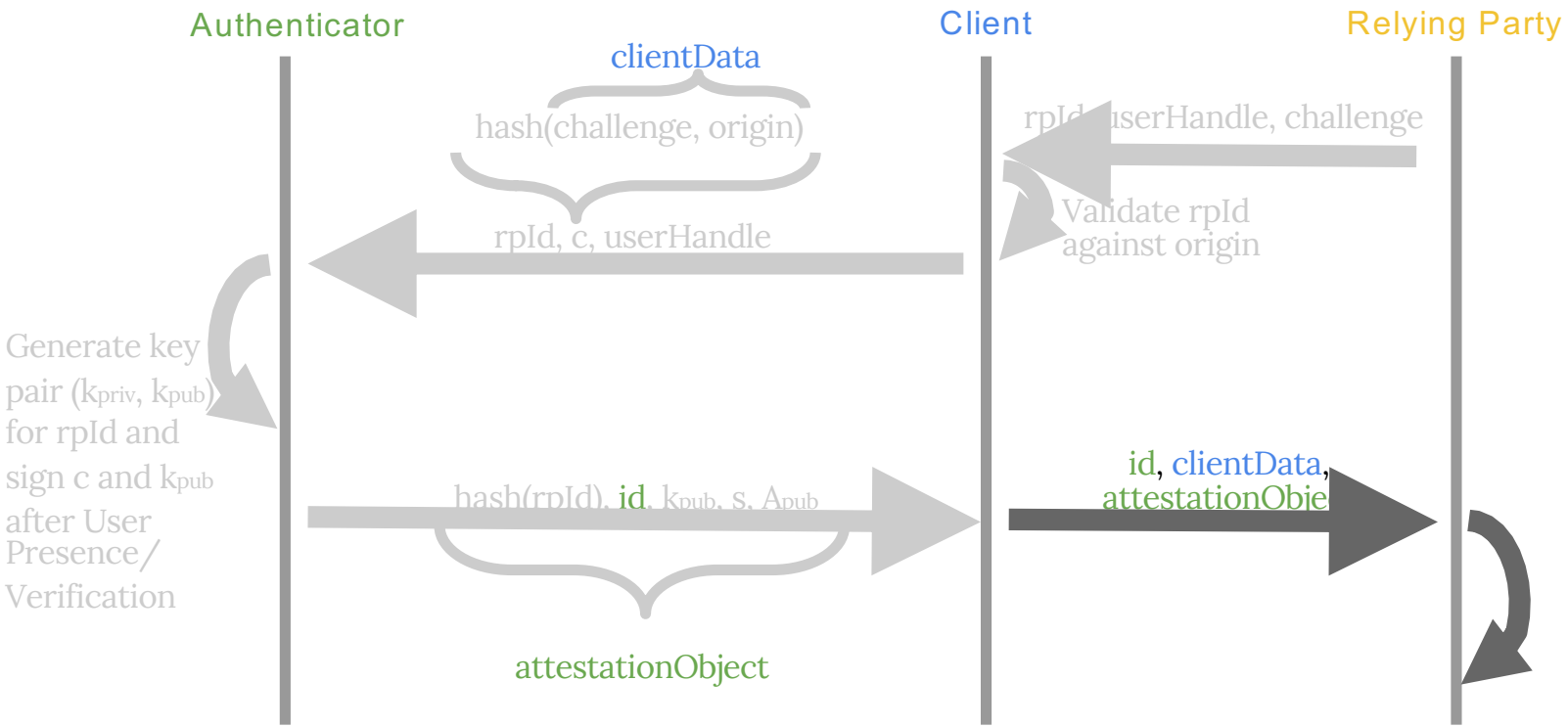
yubico

Registration Sequence



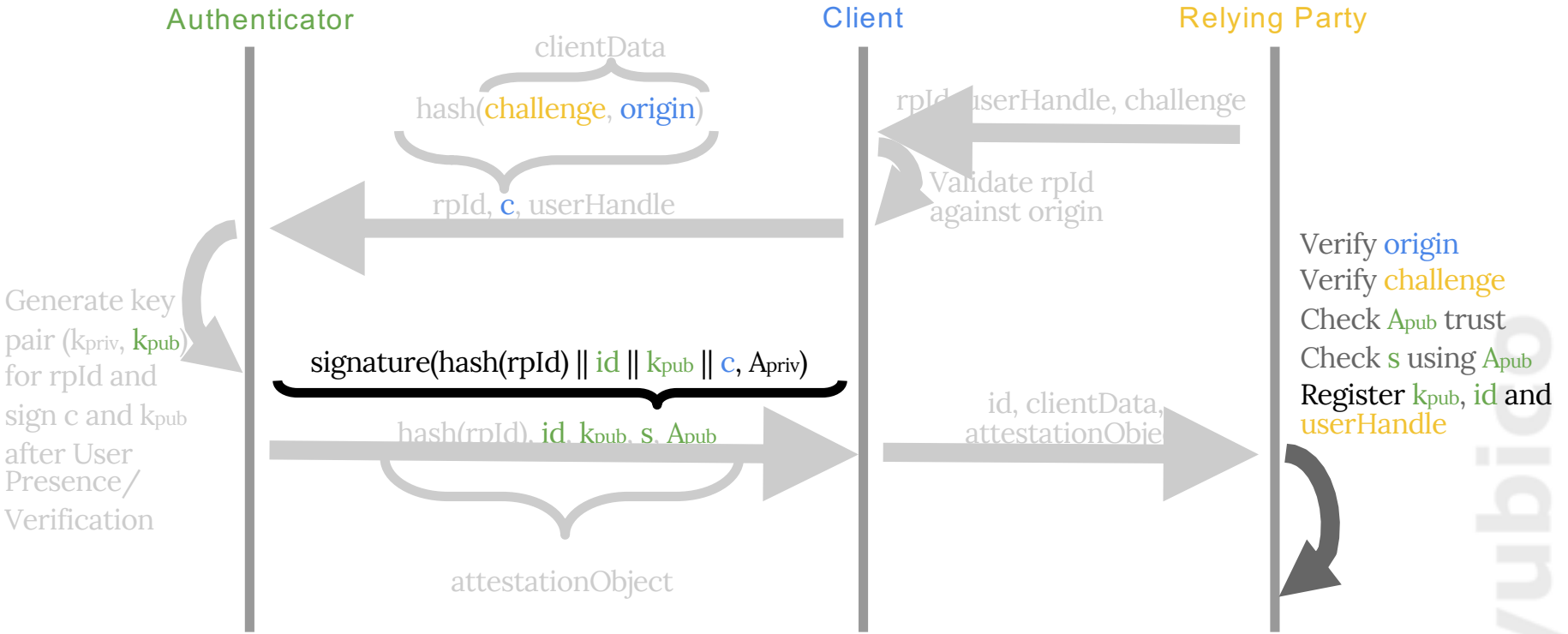
yubico

Registration Sequence



yubico

Registration Sequence



Registration Recap

1. **Relying Party generates challenge**
   - Prevents replay
2. **Client validates origin**
   - Prevents phishing
3. **Authenticator checks user presence and consent**
   - Prevents silent tracking
4. **Authenticator creates key pair**
   - No secret is shared with Relying Party
5. **Relying Party verifies attestation signature**
   - Prevents phishing
   - Proof that private key is safe

Registration API

- `navigator.credentials.create()` method: https://www.w3.org/TR/webauthn/#createCredential
- Options: https://www.w3.org/TR/webauthn/#dictdef-publickeycredentialcreationoptions

```
dictionary PublicKeyCredentialCreationOptions {
    required PublicKeyCredentialRpEntity         rp;
    required PublicKeyCredentialUserEntity       user;

    required BufferSource                              challenge;
    required sequence<PublicKeyCredentialParameters>   pubKeyCredParams;

    unsigned long                                timeout;
    sequence<PublicKeyCredentialDescriptor>      excludeCredentials = [];
    AuthenticatorSelectionCriteria               authenticatorSelection;
```

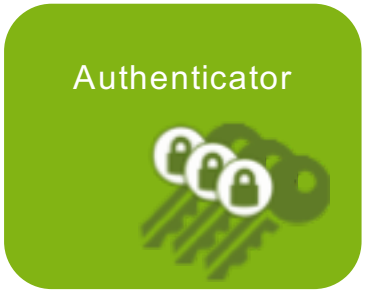
What is Attestation?

- Attestation is how authenticators prove to a Relying Party that the keys they generate originate from genuine devices with certified characteristics
- Enables passwordless, High Assurance MFA
- A chain of trust can be built to validate a genuine authenticator and establish a hardware root of trust

Requesting Attestation

- **Registration parameter:**
  `PublicKeyCredentialCreationOptions.attestation`
  - Values:
    - **“`none`”**: No signature requested
    - **“`indirect`”**: Some (proxied) signature requested
    - **“`direct`”**: Original signature requested
  - Defaults to “none”
    - Most sites unlikely to care
    - Better than plain password auth
    - No reliance on external metadata
    - Not appropriate for high assurance MFA
- **To enable: pass argument `attestation:` “`direct`”**

Resident Keys



- Authenticators store keys, user and RP Info
- Credential ID (id) returned after authentication
- First-factor roaming authentication (Tap & Go)
- Bootstrapping and Backup
- High Assurance MFA (device + PIN or biometrics)

Authentication Demo

https://demo.yubico.com/webauthn

Try it yourself:

- **Operating Systems:**
  - Windows
  - Linux
  - Mac
- **Browsers:**
  - Chrome
  - Firefox 60
  - Edge (Insider builds)
- **YubiKey Models:**
  - YubiKey NEO
  - YubiKey 4 Series (YubiKey 4, YubiKey 4 Nano, YubiKey 4C, YubiKey 4C Nano)
  - FIDO U2F Security Key
  - Security Key by Yubico

yubico

yubico

TRY WEBAUTHN

This demo will let you register and log in to an account using Web Authentication, also known as WebAuthn. This requires a WebAuthn authenticator device, as well as a browser with WebAuthn support.

Start by registering a user, then try logging in.

Register | Login | Login without username | Add authenticator | Delete account

Registration completed!

You have now completed WebAuthn registration!

Use the login form to test authentication using the registered authenticator.

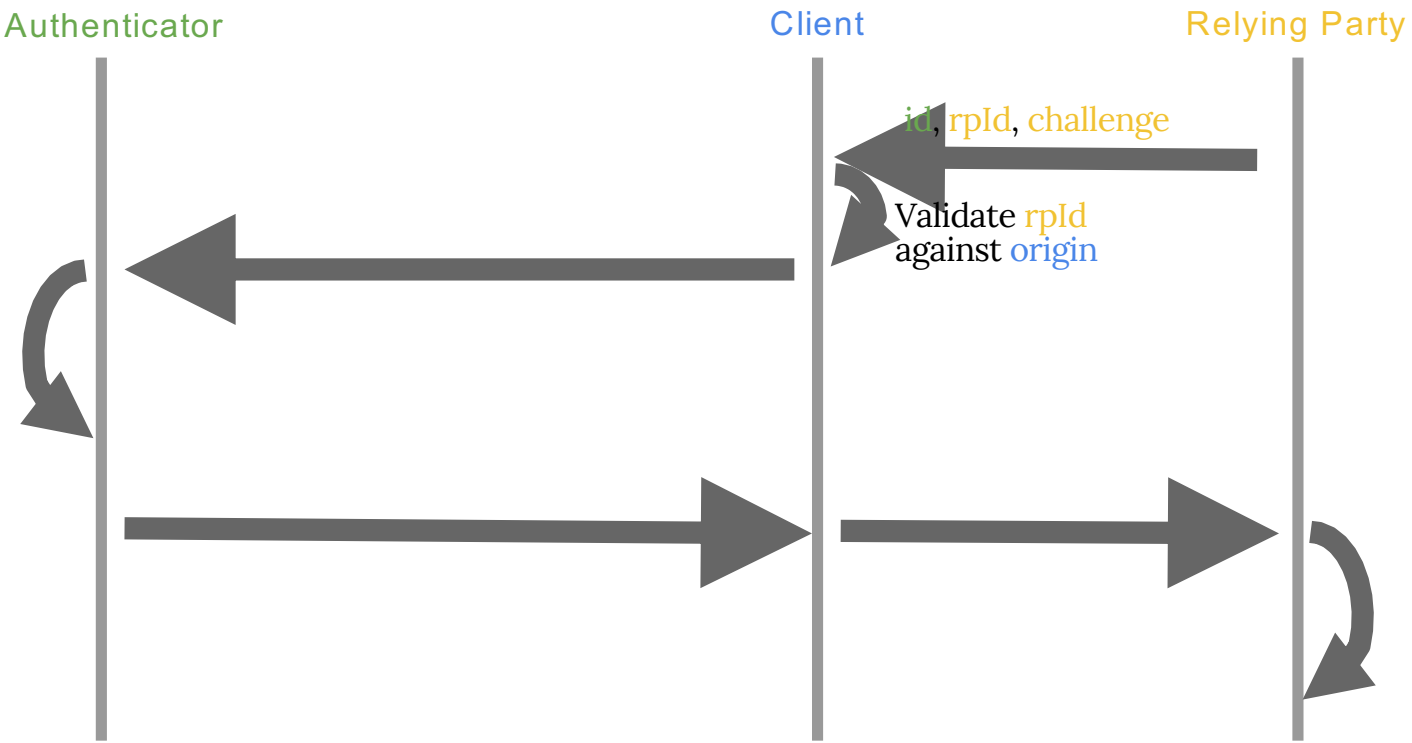
» Warning: Attestation is not trusted!

» Warning: assertion failed: Authenticator extensions {hmac-secret} are not a subset of requested extensions {}.

Unverified device
The type of device you have used is unknown to the server.

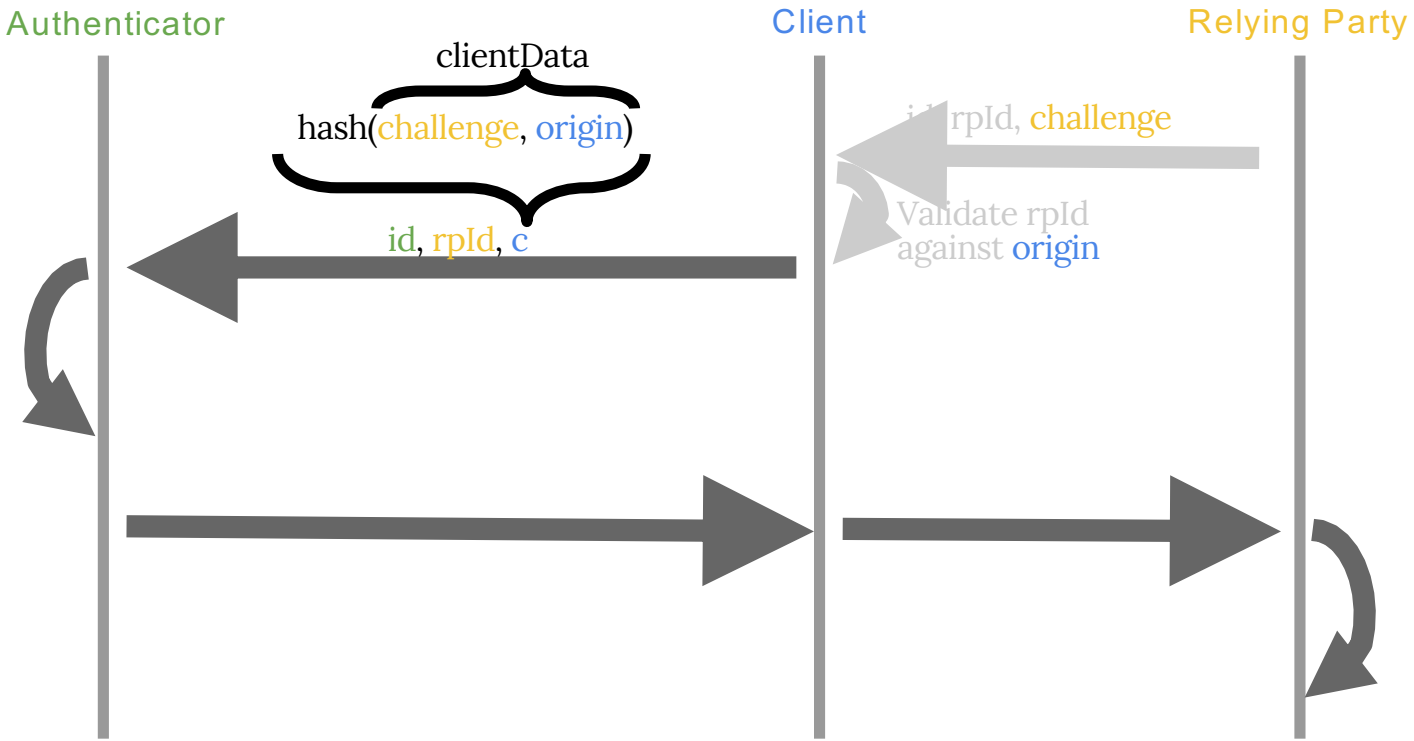
Authentication Sequence

Second-factor mode



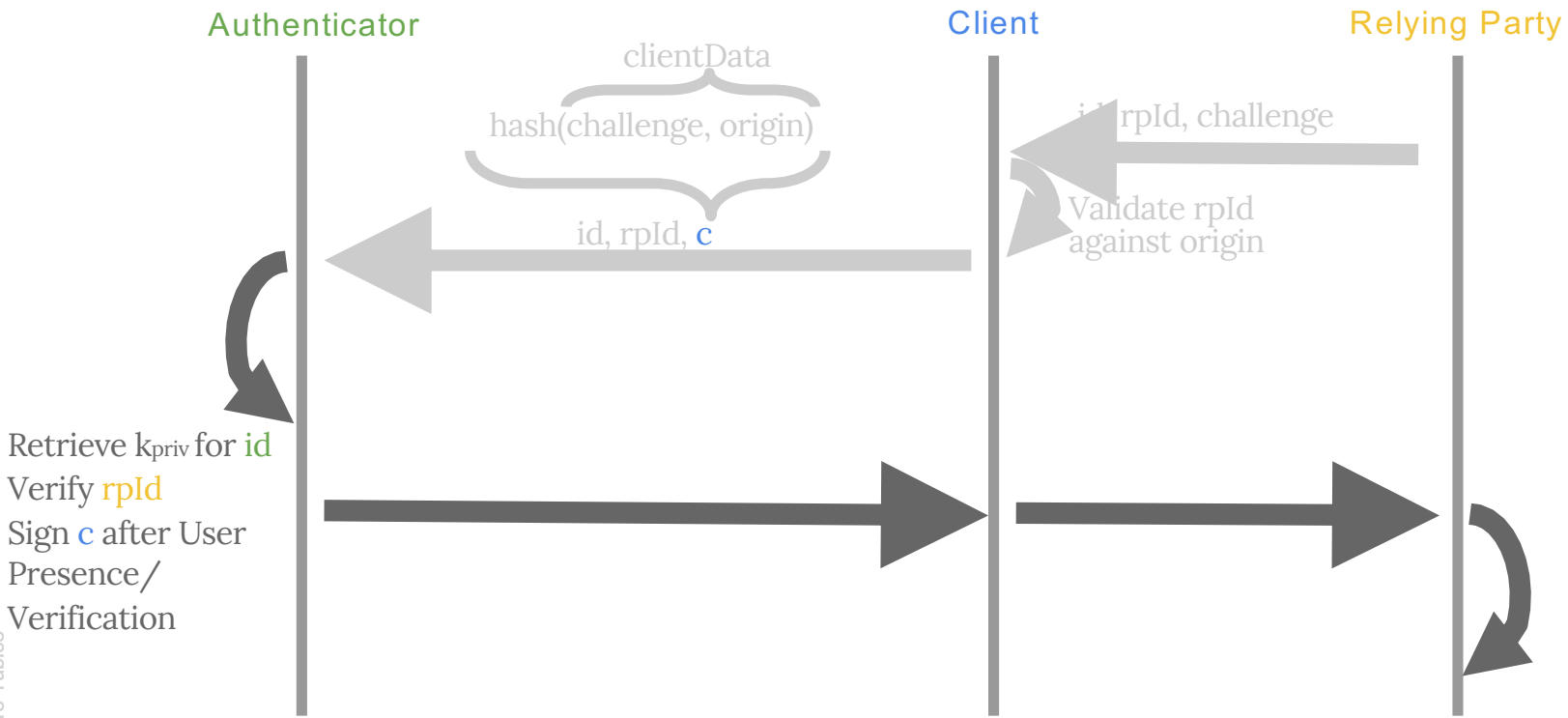
Authentication Sequence

Second-factor mode



Authentication Sequence

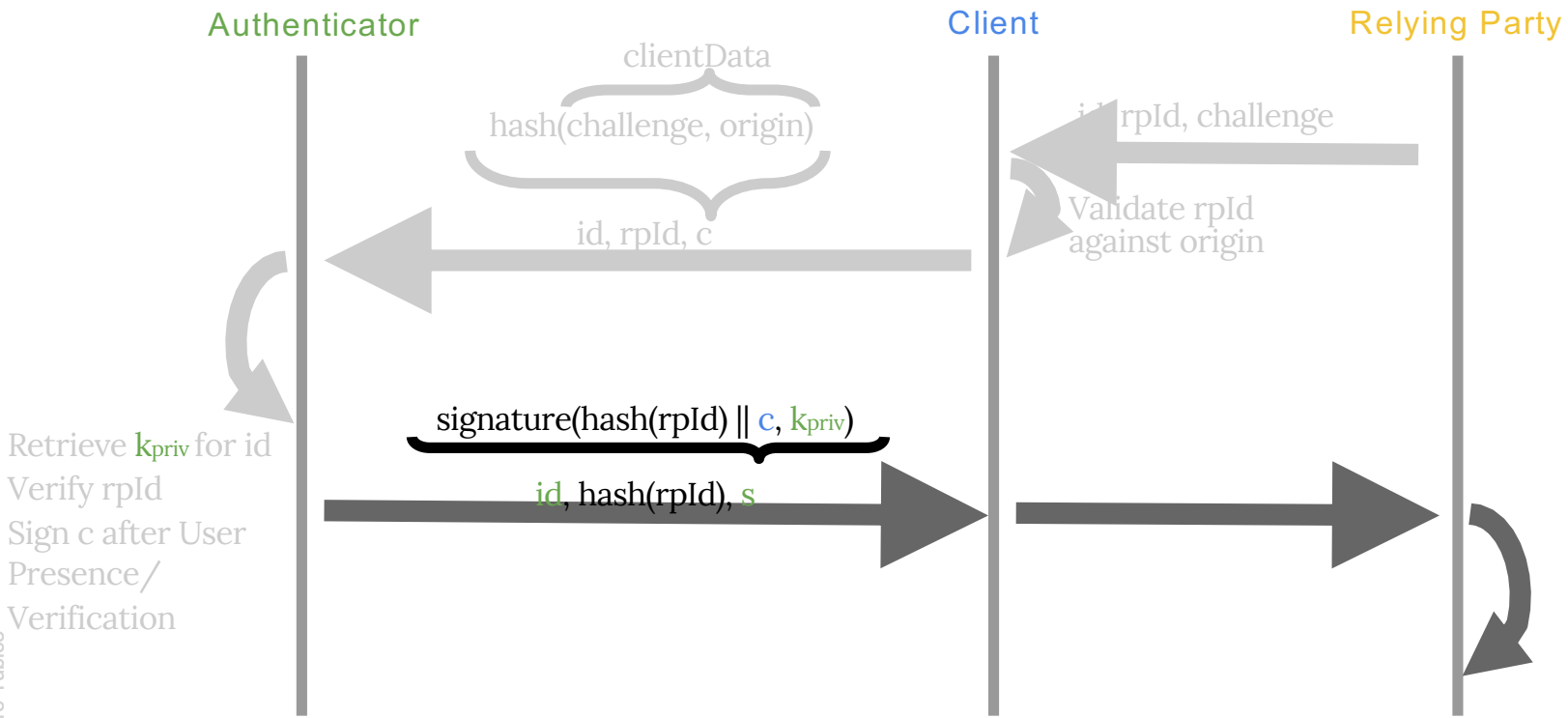
Second-factor mode



yubico

Authentication Sequence

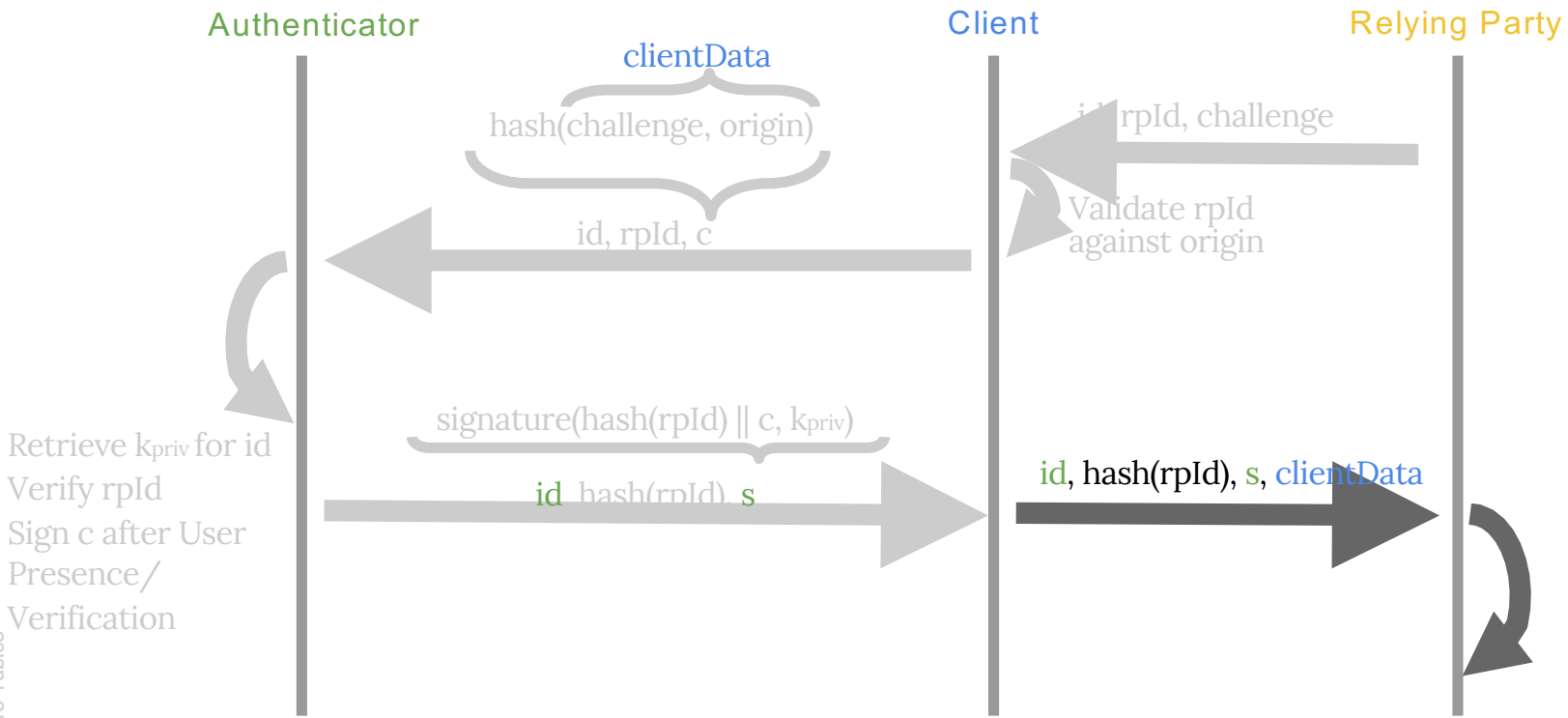
Second-factor mode



yubico

Authentication Sequence

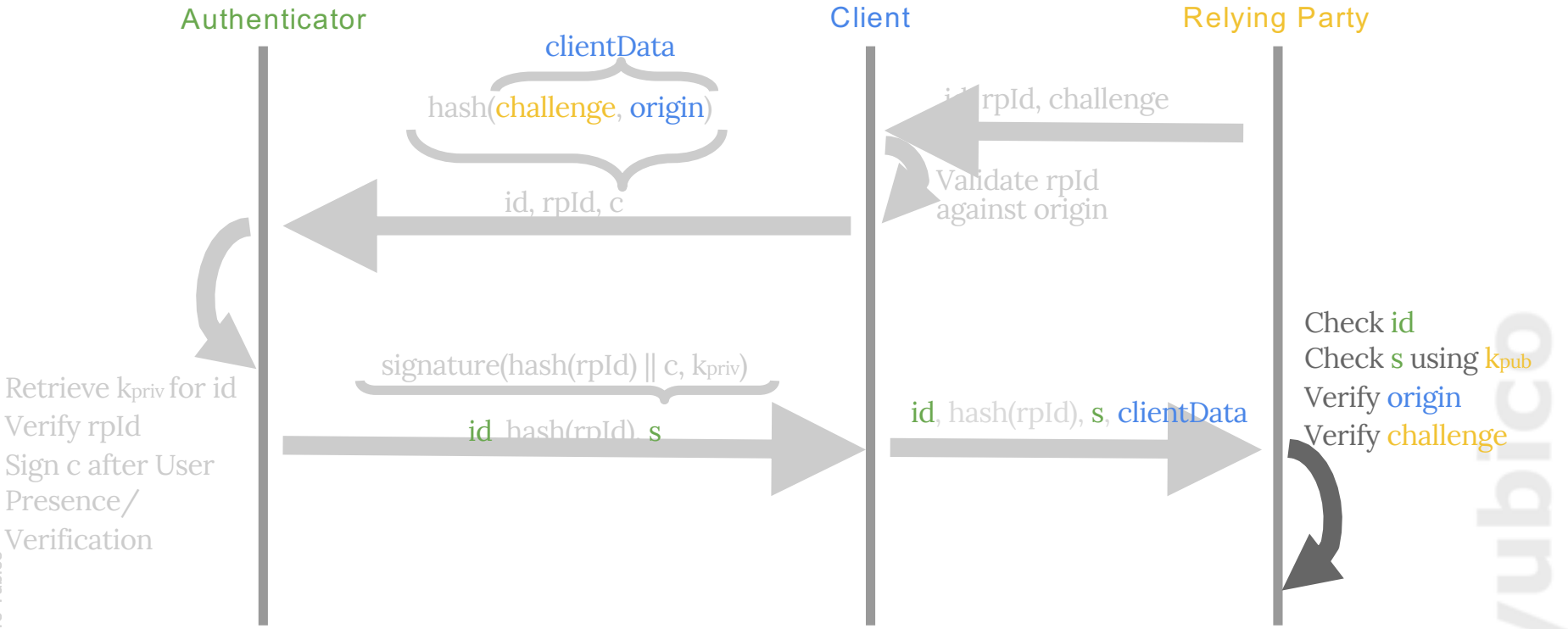
Second-factor mode



yubico

Authentication Sequence

Second-factor mode



Authentication Recap

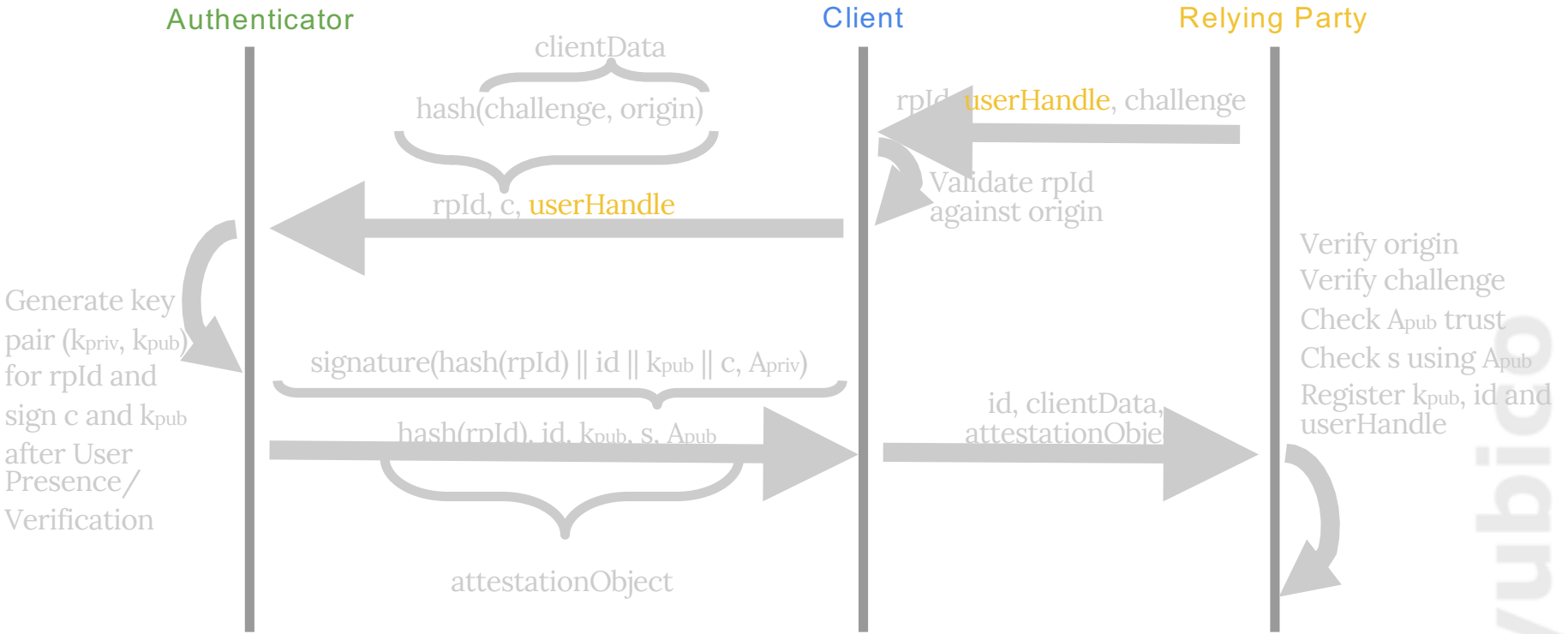
1. **Relying Party generates challenge**
   - Prevents replay
2. **Client validates origin**
   - Prevents phishing
3. **Authenticator checks user presence and consent**
   - Prevents silent tracking and covert login
4. **Authenticator checks and signs RP ID**
   - Credential is restricted to one Relying Party
5. **Relying Party verifies signature**
   - Prevents phishing
   - Prevents man-in-the-middle

Authentication Sequence

First-factor mode

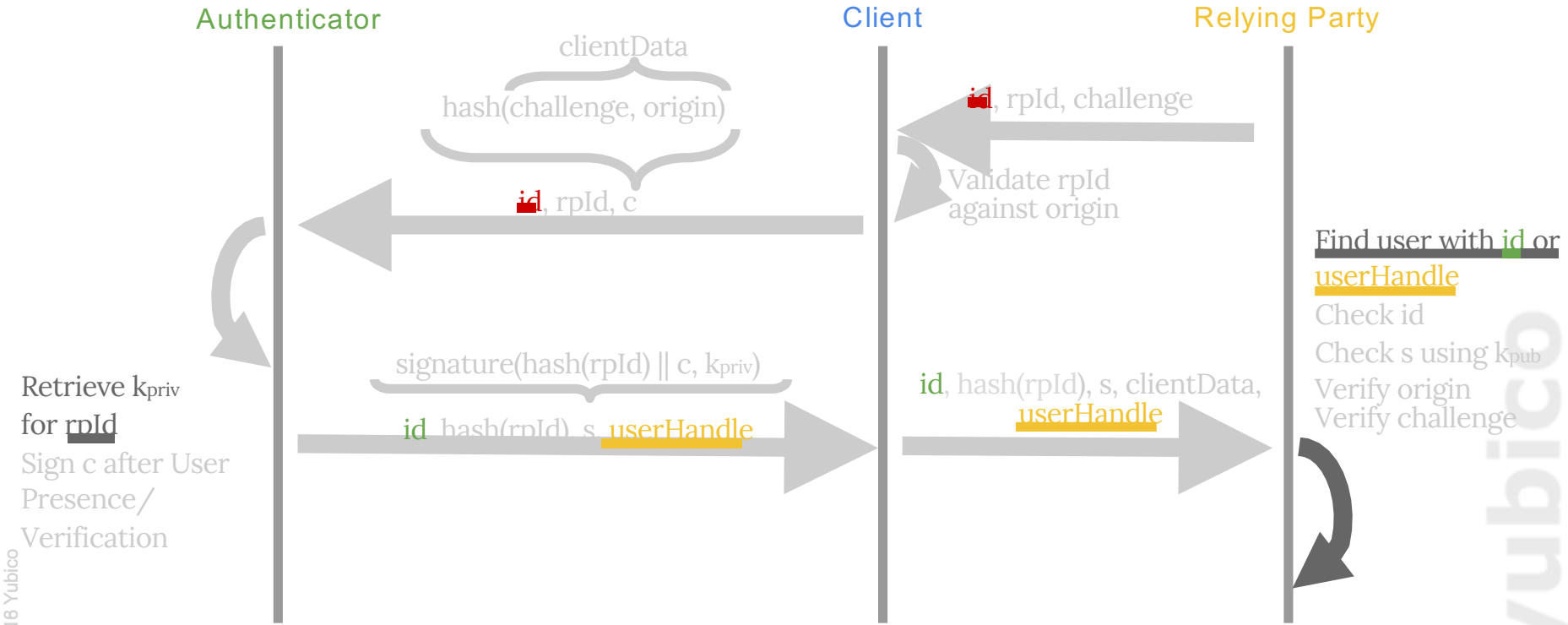
Registration Sequence

Flashback



Authentication Sequence

First-factor mode



Authentication Sequence

First-factor mode

- **Requires option `requireResidentKey: true` at registration**
- **Limited storage**
- **Combine with User Verification for MFA**
  - Set option `userVerification: “required”`
  - Requires trusted attestation

Authentication API

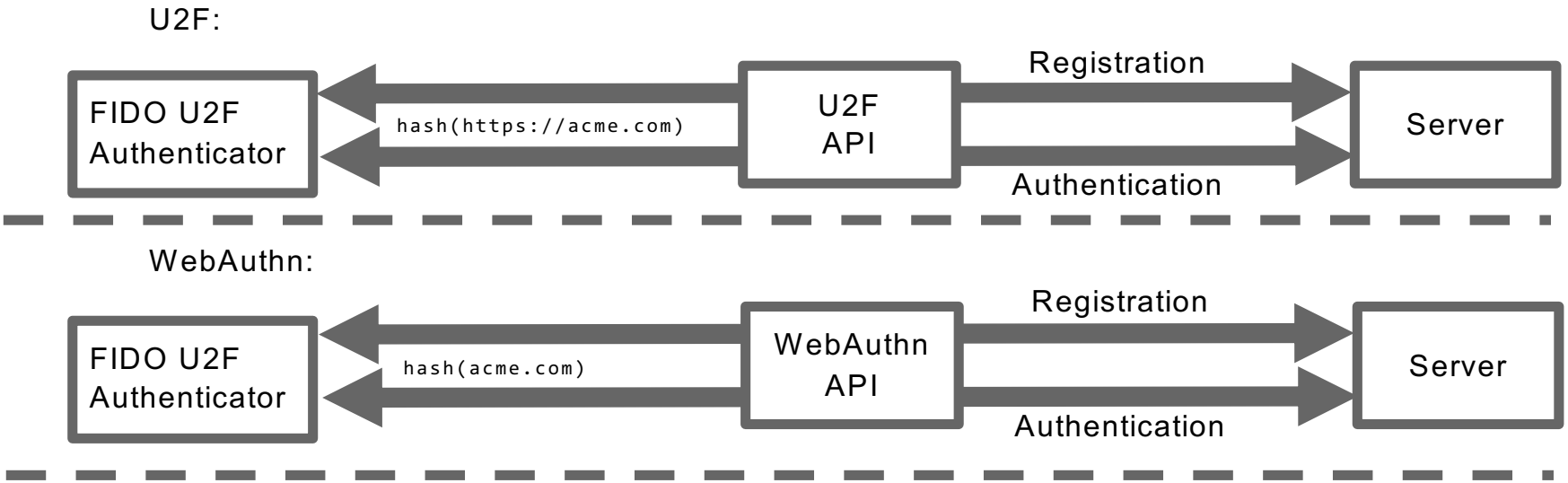
- `navigator.credentials.get()` method: https://www.w3.org/TR/webauthn/#getAssertion
- Options: https://www.w3.org/TR/webauthn/#dictdef-publickeycredentialrequestoptions

```
dictionary PublicKeyCredentialRequestOptions {
    required BufferSource                challenge;
    unsigned long                        timeout;
    USVString                            rpId;
    sequence<PublicKeyCredentialDescriptor> allowCredentials = [];
    UserVerificationRequirement          userVerification = "preferred";
    AuthenticationExtensionsClientInputs extensions;
};
```

Upgrading from U2F

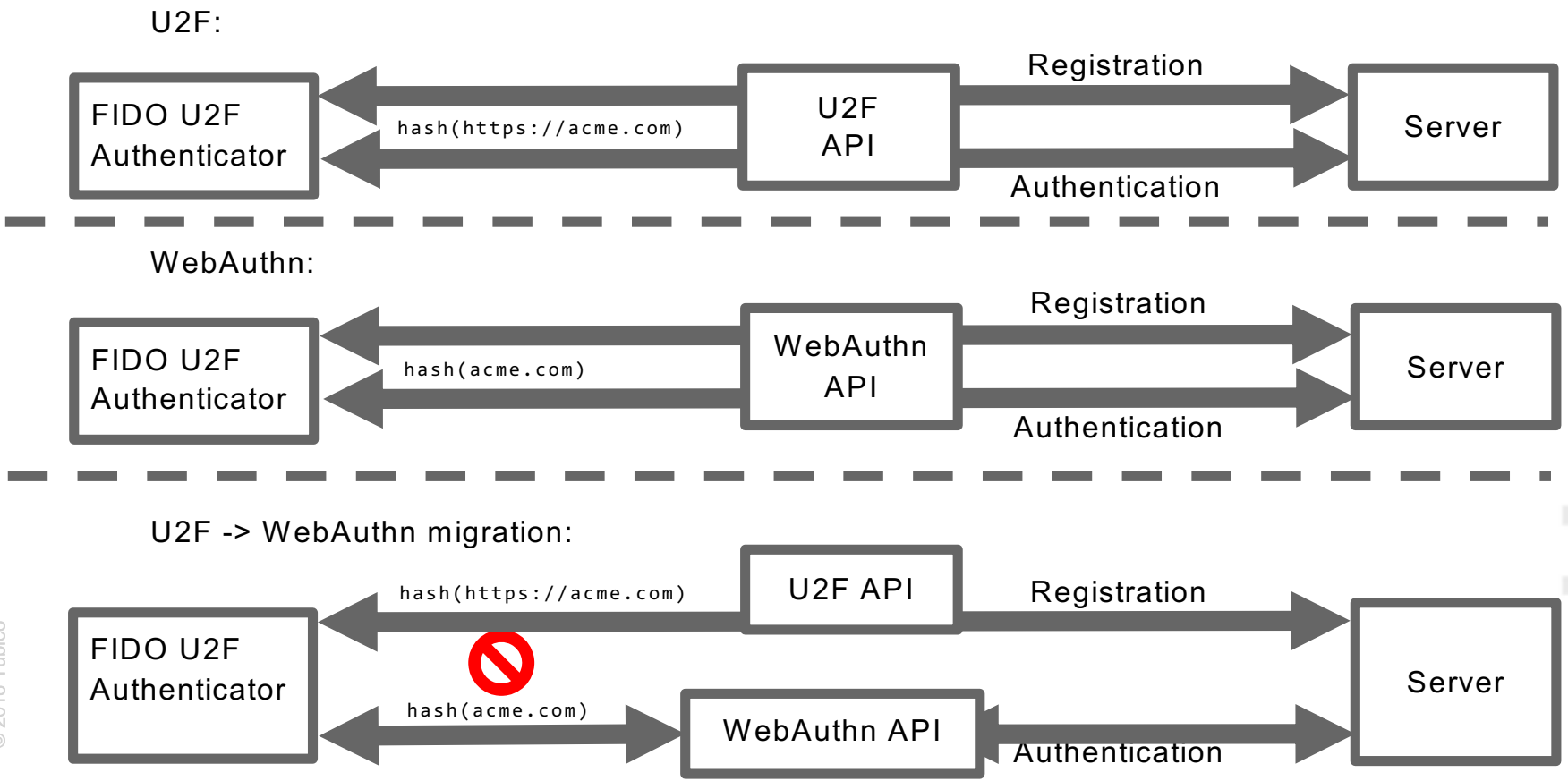
- **New registration: Works by default**
- **Existing U2F credentials:**
  - WebAuthn RP ID incompatible with U2F AppID
  - Supported via `appid` extension

Upgrading from U2F

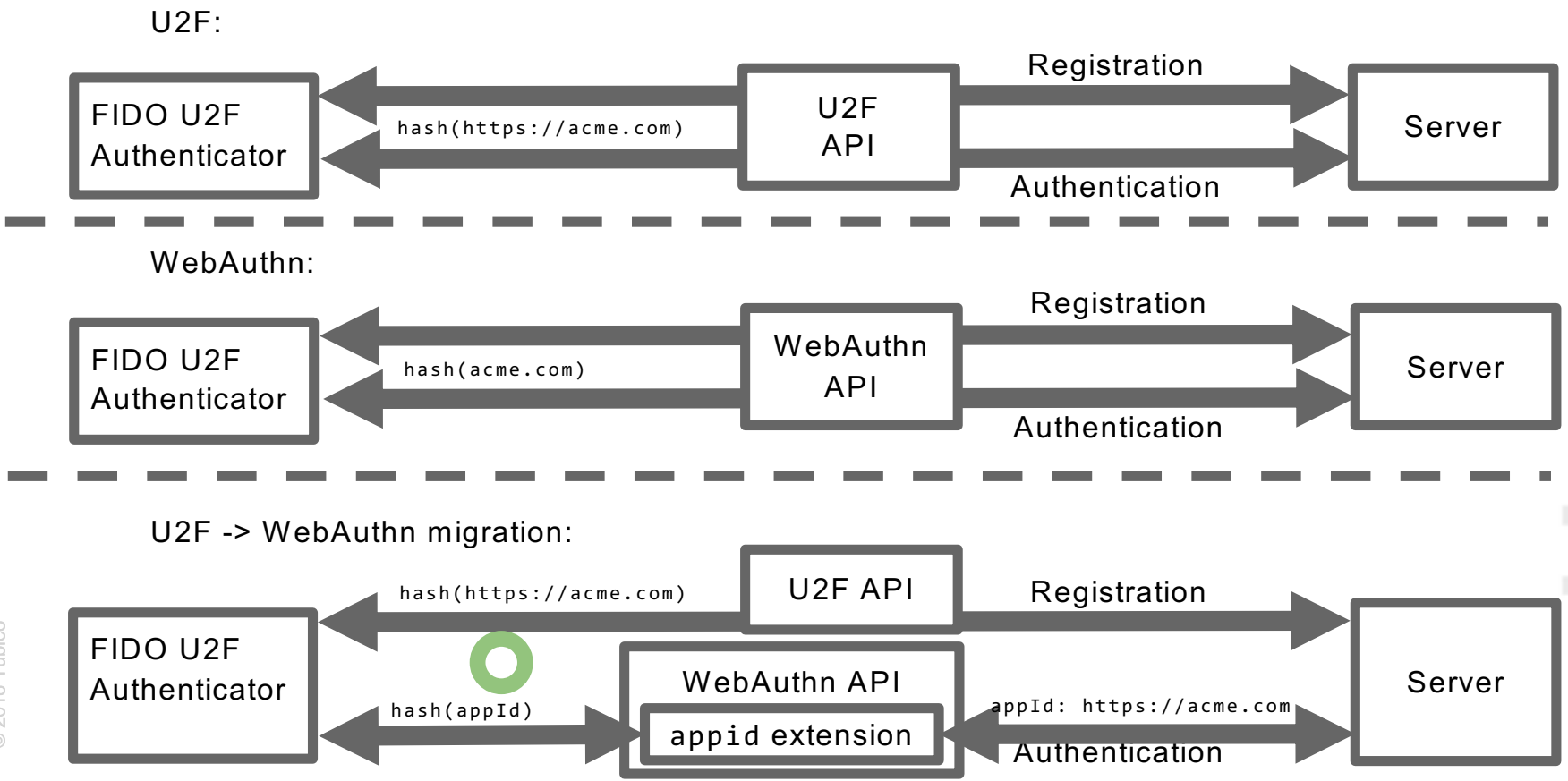


- Works by default with new registration
- WebAuthn RPID require using AppID extension to support existing U2F credentials

Upgrading from U2F



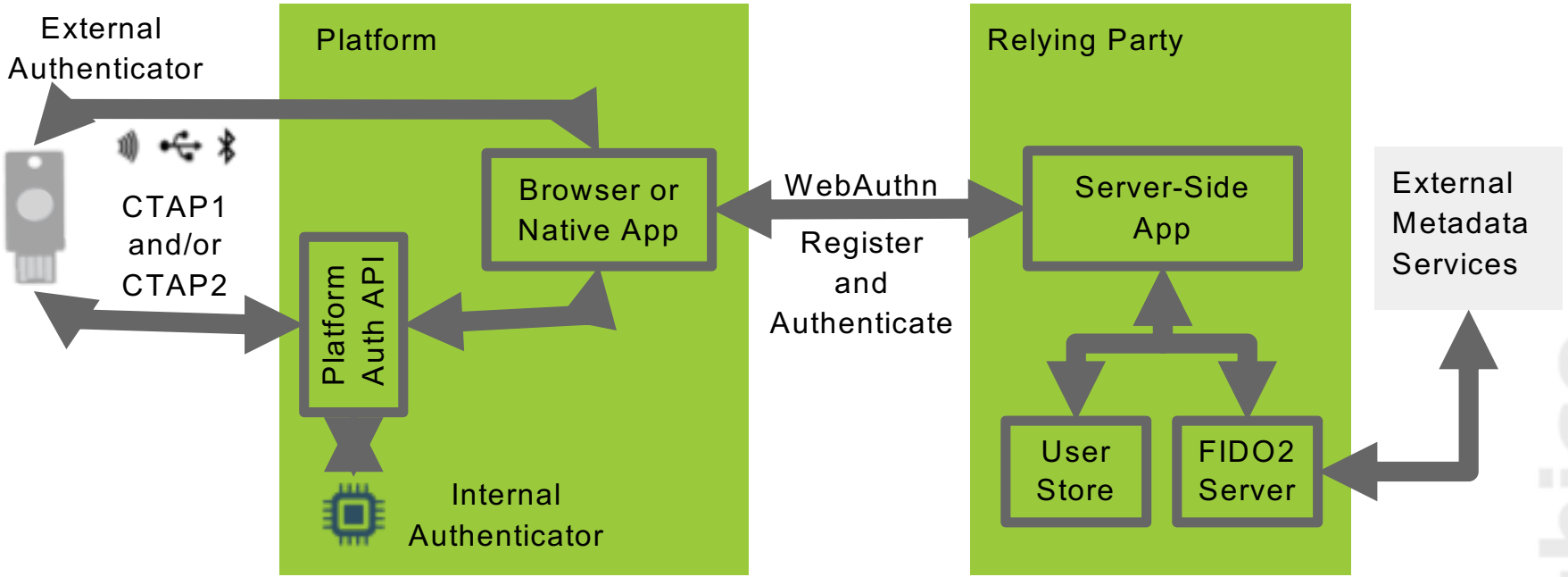
Upgrading from U2F



Deploying FIDO2

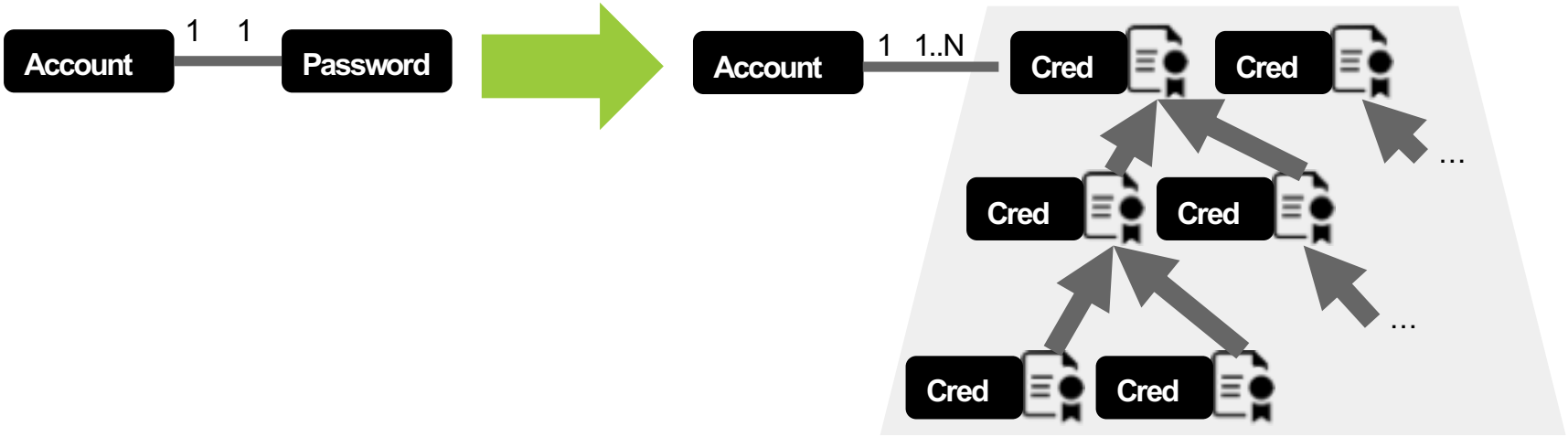
Design considerations for building first-class identity services

Application Architecture



yubico

Designing FIDO2 Credentials



Credential Registration Drives Account Recovery

Best Practices

- **Enable multiple FIDO credentials**
  - As part of the authentication flow, prompt the user to register new authenticators
- **Personalize FIDO credentials**
  - Allow users to customize metadata about authenticators
- **Record attestation information during registration process**
- **Extract device capabilities from attestation**
- **Allow approved devices by attestation**
- **Seek additional assurances from external metadata services**

Implementing WebAuthn

Demo sites:

- Microsoft: https://webauthntest.azurewebsites.net
- FIDO: https://webauthn.org/
- Yubico: https://demo.yubico.com/webauthn

Open source libraries:

- Yubico Java WebAuthn Server: https://github.com/Yubico/java-webauthn-server
- Mastercard WebAuthn Reference Implementation: https://github.com/Mastercard/fido2-rp-spring
- Yubico Python library: https://github.com/Yubico/python-fido2
- Yubico C library: https://github.com/Yubico/libfido2

References:

- Mozilla Developer Network (MDN) Web Authentication API: https://developer.mozilla.org/en-US/docs/Web/API/Web_Authentication_API

Summary

- Introduced FIDO Authentication, CTAP, and WebAuthn
- WebAuthn enables strong authentication which protects against phishing and replay attacks
- Attestation is used to build a chain of trust to verify authenticity claims
- Resident Keys provide high assurance MFA and enable passwordless authentication
- Migration path from U2F devices and credentials to WebAuthn
- Deployment considerations and best practices

Questions

yubico

Trust the Net

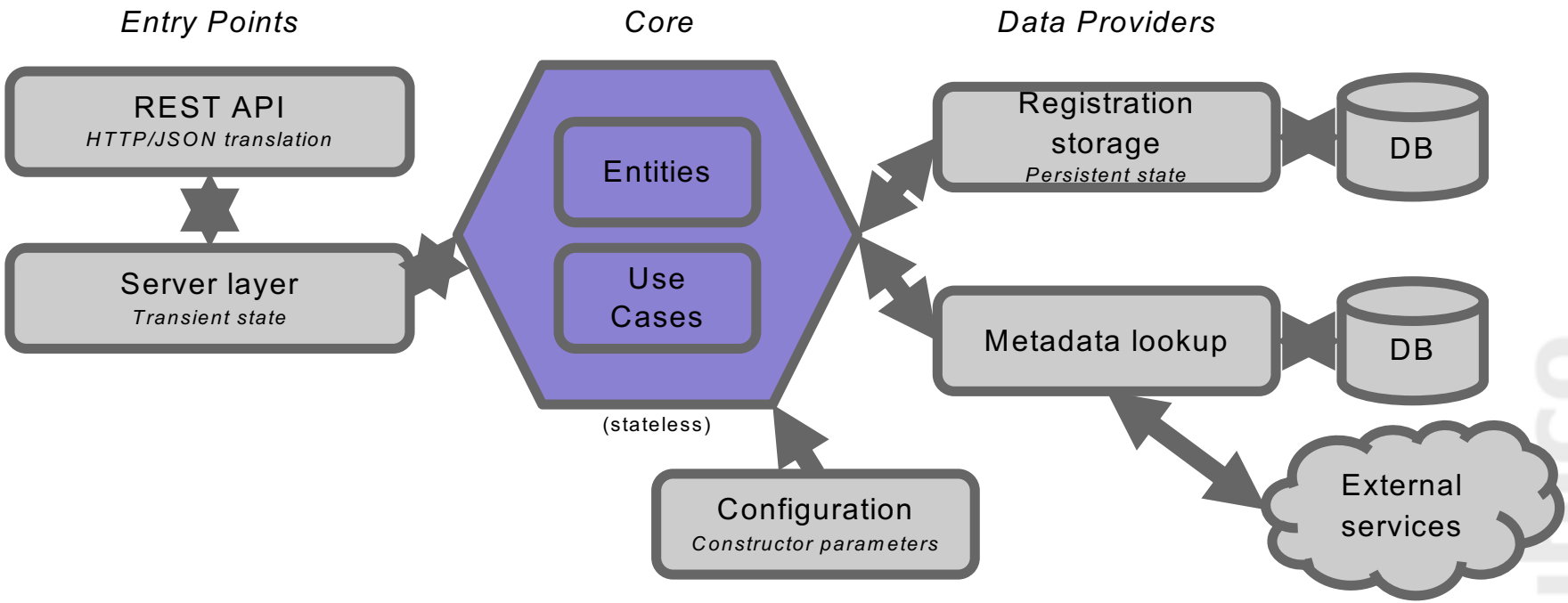
Backup Slides

Yubico/java-webauthn-server

webauthn-server-core/

- **Entity Data Model**
  - Assertion Request
  - Assertion Result
  - Attestation Object
  - Authenticator Response
  - Public Key Credential
  - Public Key Credential Creation and Request Options
  - Registration Result
  - User Identity
  - Attestation
- **Data Providers**
  - Credential Repository
  - Metadata Service
- **Use Cases**
  - Registration
  - Authentication
  - Authenticated Actions

WebAuthn Demo Server Architecture



What you need to provide

- **Storage for requests**
  - Completely external to the library
  - Library simply returns request objects
  - finish* methods expect them to be passed back in
- **Storage for credentials (registrations)**
  - Library requires an adapter object (`CredentialRepository`)
  - Library only looks credentials up
  - You need to save new registrations to the DB
- **(Optional) Additional authenticator metadata sources**
  - Library includes FIDO Metadata Service connector (in the future)
  - Optional module with Yubico device metadata as static files

Registration Walkthrough

Create Credential Request Client - Server

```
navigator.credentials.create({
    publicKey: {
        "rp": {
          "name": "Yubico Web Authentication demo",
          "id": "demo.yubico.com"
        },
        "user": {
          "name": "a.user",
          "displayName": "a.user",
          "id": "x_s5zXcAlaWEytbhHhoG4LhLX4KiImMSlnPIyEnzpQw"
        },
        "challenge": "oOJ_HV6upN3537iHDgtxeMEH2iJSpo47w_YxSK8BOHE",
        "pubKeyCredParams": [
          {
            "alg": -7,
            "type": "public-key"
          }
        ],
        "excludeCredentials": [],
        "authenticatorSelection": {
          "requireResidentKey": false,
          "userVerification": "preferred"
        },
        "attestation": "direct"
    }
});
```

```
case class PublicKeyCredentialCreationOptions(

  rp: RelyingPartyIdentity,
  user: UserIdentity,
  @JsonIgnore
  challenge: ArrayBuffer,
  pubKeyCredParams: java.util.List[PublicKeyCredentia
  timeout: Optional[Long] = None.asJava,
  excludeCredentials: Optional[java.util.Collection[
  authenticatorSelection: Optional[AuthenticatorSele
  attestation: AttestationConveyancePreference = Atte
  extensions: Optional[AuthenticationExtensionsClien

) extends JsonSerializable {

  @JsonProperty("challenge")
  def challengeBase64: String = U2fB64Encoding.encode
}
```

Registration Request

```
2   public class RegistrationRequest {
3
4       String username;
5       String credentialNickname;
6       String requestId;
7       PublicKeyCredentialCreationOptions publicKeyCredentialCreationOptions;
8
9   }
```

requestId is specific to this implementation

Start Registration

```
public Either<String, RegistrationRequest> startRegistration(String username, String displayName, String credentialNickname) {

  if (userStorage.getRegistrationsByUsername(username).isEmpty()) {
      byte[] userId = challengeGenerator.generateChallenge();

      RegistrationRequest request = new RegistrationRequest(
          username,
          credentialNickname,
          U2fB64Encoding.encode(challengeGenerator.generateChallenge()),
          rp.startRegistration(
              new UserIdentity(username, displayName, userId, Optional.empty()),
              Optional.of(userStorage.getCredentialIdsForUsername(username)),
              Optional.empty()
          )
      );
      registerRequestStorage.put(request.getRequestId(), request);
      return new Right(request);
  } else {
      return new Left("The username \"" + username + "\" is already registered.");
  }
}
```

Start Registration

```
public Either<String, RegistrationRequest> startRegistration(String username, String displayName, String credentialNickname) {

  if (userStorage.getRegistrationsByUsername(username).isEmpty()) {
      byte[] userId = challengeGenerator.generateChallenge();

      RegistrationRequest request = new RegistrationRequest(
          username,
          credentialNickname,
          U2fB64Encoding.encode(challengeGenerator.generateChallenge()),
          rp.startRegistration(
              new UserIdentity(username, displayName, userId, Optional.empty()),
              Optional.of(userStorage.getCredentialIdsForUsername(username)),
              Optional.empty()
          )
      );
      registerRequestStorage.put(request.getRequestId(), request);
      return new Right(request);
  } else {
      return new Left("The username \"" + username + "\" is already registered.");
  }
}
```

Nickname is specific to this implementation

Start Registration

```
public Either<String, RegistrationRequest> startRegistration(String username, String displayName, String credentialNickname) {

  if (userStorage.getRegistrationsByUsername(username).isEmpty()) {
      byte[] userId = challengeGenerator.generateChallenge();

      RegistrationRequest request = new RegistrationRequest(
          username,
          credentialNickname,
          U2fB64Encoding.encode(challengeGenerator.generateChallenge()),
          rp.startRegistration(
              new UserIdentity(username, displayName, userId, Optional.empty()),
              Optional.of(userStorage.getCredentialIdsForUsername(username)),
              Optional.empty()
          )
      );
      registerRequestStorage.put(request.getRequestId(), request);
      return new Right(request);
  } else {
      return new Left("The username \"" + username + "\" is already registered.");
  }
}
```

The userId is randomly generated

Start Registration

```
public Either<String, RegistrationRequest> startRegistration(String username, String displayName, String credentialNickname) {

  if (userStorage.getRegistrationsByUsername(username).isEmpty()) {
      byte[] userId = challengeGenerator.generateChallenge();

      RegistrationRequest request = new RegistrationRequest(
          username,
          credentialNickname,
          U2fB64Encoding.encode(challengeGen                erateChallenge()),
          rp.startRegistration(
              new UserIdentity(username, displayName, userId, Optional.empty()),
              Optional.of(userStorage.getCredentialIdsForUsername(username)),
              Optional.empty()
          )
      );
      registerRequestStorage.put(request.getRequestId(), request);
      return new Right(request);
  } else {
      return new Left("The username \"" + username + "\" is already registered.");
  }
}
```

Rp is an instance of the Relying Party object from the core library and initiates the register use case

Start Registration

```
public Either<String, RegistrationRequest> startRegistration(String username, String displayName, String credentialNickname) {

  if (userStorage.getRegistrationsByUsername(username).isEmpty()) {
      byte[] userId = challengeGenerator.generateChallenge();

      RegistrationRequest request = new RegistrationRequest(
          username,
          credentialNickname,
          U2fB64Encoding.encode(challengeGenerator.generateChallenge()),
          rp.startRegistration(
              new UserIdentity(username, displayName, userId, Optional.empty()),
              Optional.of(userStorage.getCredentialIdsForUsername(username)),
              Optional.empty()
          )
      );
      registerRequestStorage.put(request.getRequestId(), request);
      return new Right(request);
  } else {
      return new Left("The username \"" + username + "\" is already registered.");
  }
}
```

The registration request is temporarily stored, to be completed later

Client - Create Credential Response

```
// authenticatorAttestationResponse
{
    "id": "RvPR8sycnQMH32jNKtxA_wKuqMAvJILXFE9vgDjzSHC5Ez5qhY5vYTKsPbx9FbnNSdSKpNxkb4bYQWq5eWr9xA",
    "response": {
      "attestationObject": "o2NmbXRoZmlkby11MmZnYXR0U3RtdKJjc2lnWEYwRAIgV9kXUC-T2cvbj-BjSKWe...",
      "clientDataJSON": "eyJjaGFsbGVuZ2UiOiJvT0pfSFY2dXBOMzUzN2lIRGd0eGVNRUgyaUpTcG80N3dfWXh..."
    },
    "clientExtensionResults": {}
}
```

```
//clientDataJSON
{
    "challenge": "oOJ_HV6upN3537iHDgtxeMEH2iJSpo47w_YxSK8BOHE",
    "new_keys_may_be_added_here": "do not compare clientDataJSON against a template. See https://goo.gl/yabPex",
    "origin": "https://demo.yubico.com",
    "tokenBinding": {
      "status": "not-supported"
    },
    "type": "webauthn.create"
}
```

Client - Create Credential Response

```
// authenticatorAttestationResponse
{
    "id": "RvPR8sycnQMH32jNKtxA_wKuqMAvJILXFE9vgDjzSHC5Ez5qhY5vYTKsPbx9FbnNSdSKpNxkb4bYQWq5eWr9xA",
    "response": {
      "attestationObject": "o2NmbXRoZmlkby11MmZnYXR0U3RtdKJjc2lnWEYwRAIgV9kXUC-T2cvbj-BjSKWe...",
      "clientDataJSON": "eyJjaGFsbGVuZ2UiOiJvT0pfSFY2dXBOMzUzN2lIRGd0eGVNRUgyaUpTcG80N3dfWXh..."
    },
    "clientExtensionResults": {}
}
```

Added by Chrome

```
//clientDataJSON
{
    "challenge": "oOJ_HV6upN3537iHDgtxeMEH2iJSpo47w_YxSK8BOHE",
    "new_keys_may_be_added_here": "do not compare clientDataJSON against a template. See https://goo.gl/yabPex",
    "origin": "https://demo.yubico.com",
    "tokenBinding": {
      "status": "not-supported"
    },
    "type": "webauthn.create"
}
```

Client - Create Credential Response

```
// authenticatorAttestationResponse
{
    "id": "RvPR8sycnQMH32jNKtxA_wKuqMAvJILXFE9vgDjzSHC5Ez5qhY5vYTKsPbx9FbnN5dSKpNxkb4bYQWq5eWr9xA",
    "response": {
      "attestationObject": "o2NmbXRoZmlkby11MmZnYXR0U3RtdKJjc2lnWEYwRAIgV9kXUC-T2cvbj-BjSKWe...",
      "clientDataJSON": "eyJjaGFsbGVuZ2UiOiJvT0pfSFY2dXBOMzUzN2lIRGd0eGVNRUgyaUpTcG80N3dfWXh..."
    },
    "clientExtensionResults": {}
}
```

```
//clientDataJSON
{
    "challenge": "oOJ_HV6upN3537iHDgtxeMEH2iJSpo47w_YxSK8BOHE",
    "new_keys_may_be_added_here": "do not compare clientDat[illegible]gainst a template. See https://goo.gl/yabPex",
    "origin": "https://demo.yubico.com",
    "tokenBinding": {
      "status": "not-supported"
    },
    "type": "webauthn.create"
}
```

Will be removed in a future WebAuthn release. Valid values will be “supported” and “present”

Finish Registration

```
2   public Either<List<String>, SuccessfulRegistrationResult> finishRegistration(String responseJson) {
3
4     RegistrationResponse response = jsonMapper.readValue(responseJson, RegistrationResponse.class);
5
6     RegistrationRequest request = registerRequestStorage.getIfPresent(response.getRequestId());
7     registerRequestStorage.invalidate(response.getRequestId());
8
9         Try<RegistrationResult> registrationTry = rp.finishRegistration(
10            request.getPublicKeyCredentialCreationOptions(),
11            response.getCredential(),
12            Optional.empty()
13        );
```

Finish Registration

```
15      if (registrationTry.isSuccess()) {
16          return Right.apply(
17              new SuccessfulRegistrationResult(
18                  request,
19                  response,
20                  addRegistration(
21                      request.getUsername(),
22                      request.getPublicKeyCredentialCreationOptions().user(),
23                      request.getCredentialNickname(),
24                      response,
25                      registrationTry.get()
26                  ),
27                  registrationTry.get().attestationTrusted()
28              )
29          );
30      } else {
31          return Left.apply(Arrays.asList("Registration failed!", registrationTry.failed().get().getMessage()));
32      }
```

Adds user and credential to the credential repository

Authentication Walkthrough

Get Credential Request Client - Server

```
navigator.credentials.get({
  publicKey: {
    "challenge": "Z37e0ba2cSru6mu43RDe78dAh
    "rpId": "demo.yubico.com",
    "allowCredentials": [
      {
        "type": "public-key",
        "id": "RvPR8sycnQMH32jNKtxA_wKuqMAv.
      }
    ],
    "userVerification": "preferred"
  }
});
```

```
case class PublicKeyCredentialRequestOptions(
  @JsonIgnore
  challenge: ArrayBuffer,
  timeout: Optional[Long] = None.asJava,
  rpId: Optional[String] = None.asJava,
  allowCredentials: Optional[java.util.List[PublicKe
  userVerification: UserVerificationRequirement = Us
  extensions: Optional[AuthenticationExtensionsClien
) {
  @JsonProperty("challenge")
  def challengeBase64: String = U2fB64Encoding.encod
}
```

Assertion Request

```
2   case class AssertionRequest(
3     requestId: String,
4     username: Optional[String],
5     publicKeyCredentialRequestOptions: PublicKeyCredentialRequestOptions
6   )
```

requestId is specific to this implementation

Start Authentication

```
2  public AssertionRequest startAuthentication(Optional<String> username) {
3
4      AssertionRequest request = new AssertionRequest(
5          username,
6          U2fB64Encoding.encode(challengeGenerator.generateChallenge()),
7          rp.startAssertion(
8              username.map(userStorage::getCredentialIdsForUsername),
9              Optional.empty()
10         )
11     );
12
13     assertRequestStorage.put(request.getRequestId(), request);
14
15     return request;
16 }
```

Start Authentication

```
2  public AssertionRequest startAuthentication(Optional<String> username) {
3
4      AssertionRequest request = new AssertionRequest(
5          username,
6          U2fB64Encoding.encode(challengeGenerator.gen
7          rp.startAssertion(
8              username.map(userStorage::getCredentialIdsForUsername),
9              Optional.empty()
10         )
11     );
12
13     assertRequestStorage.put(request.getRequestId(), request);
14
15     return request;
16 }
```

Empty username means empty allowCredentials, which means only resident keys can be used

Client - Get Credential Response

```
// authenticatorAssertionResponse
{
  "id": "RvPR8sycnQMH32jNKtxA_wKuqMAvJILXFE9vgDjzSHC5Ez5qhY5vYTKsPbx9FbnNSdSKpNxkb4bYQWq5eWr9xA",
  "response": {
    "authenticatorData": "xGzvgq0bVGR3WR0Aiwh1nsPm0uy085R0v-ppaZJdA7cBAAAAEA",
    "clientDataJSON": "eyJjaGFsbGVuZ2UiOiJaMzdlMGJhMmNTcnU2bXU0M1JEZTc4ZEFoZ2JVVGhSVV81bER...",
    "signature": "MEUCIQDjRKF0AiI7sLKM0Q0nM_IkRmMgY-ysK_aPcu4F1U1yWwIgRR46E_zp5-SuGytla72F..."
  },
  "clientExtensionResults": {
    "appid": false
  }
}
```

```
//clientDataJSON
{
  "challenge": "Z37e0ba2cSru6mu43RDe78dAhgbUThRU_5lDLVe7HZg",
  "new_keys_may_be_added_here": "do not compare clientDataJSON against a template. See https://goo.gl/yabPex",
  "origin": "https://demo.yubico.com",
  "tokenBinding": {
    "status": "not-supported"
  },
  "type": "webauthn.get"
}
```

Finish Authentication

```
public Either<List<String>, SuccessfulAuthenticationResult> finishAuthentication(String responseJson) {

    final AssertionResponse response = jsonMapper.readValue(responseJson, AssertionResponse.class);

    AssertionRequest request = assertRequestStorage.getIfPresent(response.getRequestId());
    assertRequestStorage.invalidate(response.getRequestId());

    Optional<String> returnedUserHandle = Optional.ofNullable(response.getCredential().response().userHandleBase64());

    final String username;
    if (request.getUsername().isPresent()) {
        username = request.getUsername().get();
    } else {
        username = userStorage.getUsername(returnedUserHandle.get()).orElse(null);
    }

    final String userHandle = returnedUserHandle.orElseGet(() ->
        username == null
            ? null
            : userStorage.getUserHandle(username)
                .map(BinaryUtil::toBase64)
                .orElse(null)
    );
```

Server - Finish Authentication

```
Try<AssertionResult> assertionTry = rp.finishAssertion(
    request.getPublicKeyCredentialRequestOptions(),
    response.getCredential(),
    () -> userHandle,
    Optional.empty()
);

if (assertionTry.isSuccess()) {
    final AssertionResult result = assertionTry.get();

    if (result.success()) {
        userStorage.updateSignatureCountForUsername(
            username,
            response.getCredential().id(),
            result.signatureCount()
        );

        return Right.apply(
            new SuccessfulAuthenticationResult(
                request,
                response,
                userStorage.getRegistrationsByUsername(username)
            )
        );
    }
} else {
    return Left.apply(Arrays.asList("Assertion failed!",
      assertionTry.failed().get().getMessage()));
}
```

Best Practices

- **Store the verbatim attestation object**
  - Enables future re-evaluation of trust
- **Allow registering more than one credential per account**
  - Consider allowing credential nicknames
- **Weigh pros vs cons of requiring attestation**
  - Pros:
    - Higher assurance
  - Cons:
    - Maintenance for metadata store
    - Compatibility issues for unknown/new authenticators(not in metadata store)
- **Security and Privacy Considerations**
  - Refer to W3C Web Authentication API spec https://www.w3.org/TR/webauthn/#security-considerations