Passwordless (passkey) with Ruby/WebAuthn Implementation trends

Corporate Design Department
Security Group Keiko Itakura

Born in Izumo City, Shimane Prefecture

2005 **Unisys** General Technology Research Institute
System Engineer

2008 **Microsoft** Consulting Services
technical consultant

2015 **IBM** Security Division
Manager

2019 **Rakuten Group** Ecosystem Service Department
Principal Information Security Specialist /
FIDO Alliance
Japan Leadership Group/Vice Chair

2023 **Medley**
Corporate Design Department
Company-wide information security promotion officer

MEDLEY

Introduce the latest trends related to passwordless authentication (passkey) from a Ruby implementation perspective.

MEDLEY

1 History and issues of authentication methods

2 The release of synced passkeys and related technologies

3 Passkey implementation with Ruby

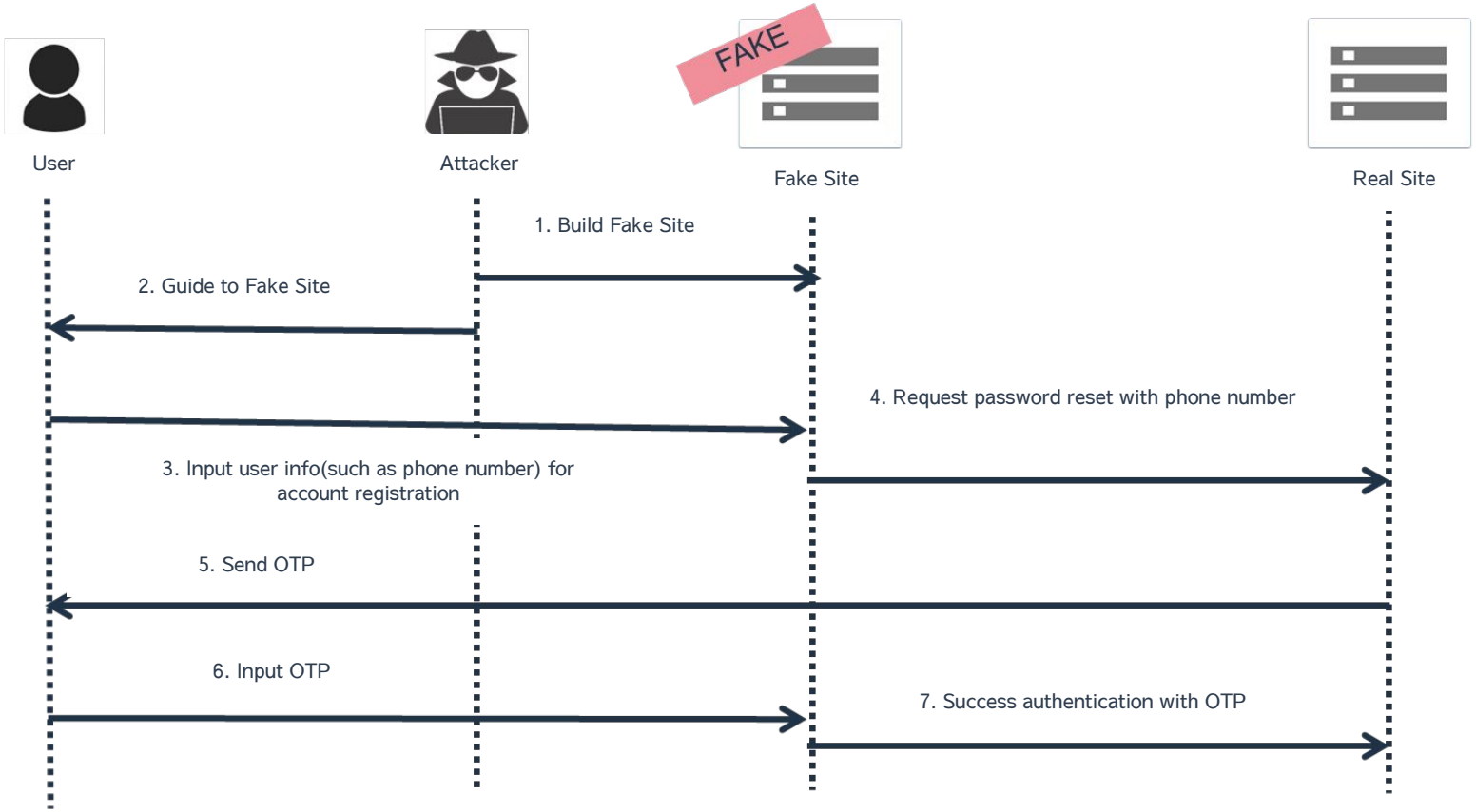
Challenges in password authentication

On the Internet, nobody knows you're a dog

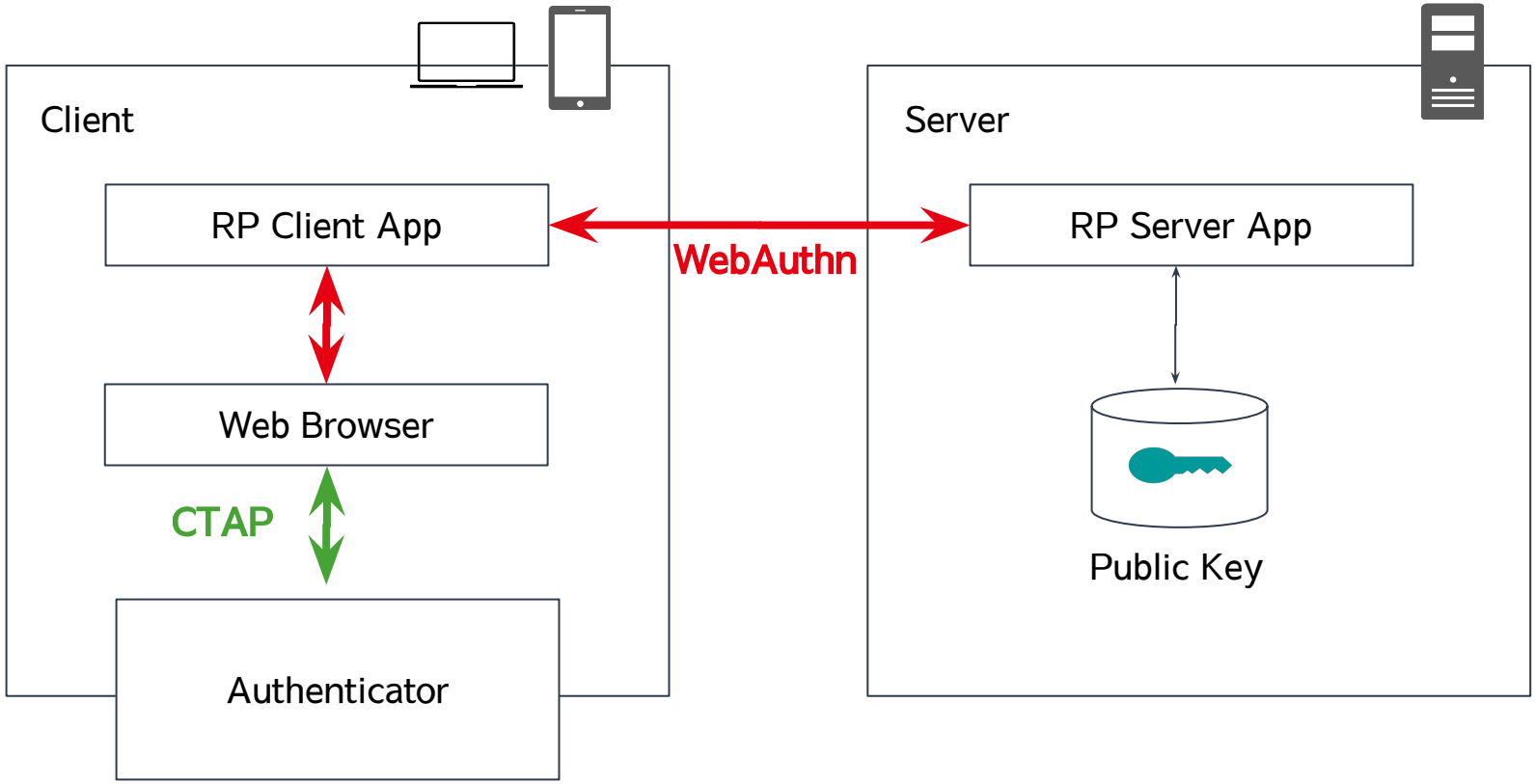
- Can't remember passwords
  - Setting a simple password that is easy to remember
  - Reusing passwords
- Many security incidents occur due to account takeover.
- Responding to inquiries due to forgotten passwords and incurring password reset operation costs

Challenges with multi-factor authentication

There have also been reports of attacks that cannot be prevented even with multi-factor authentication, such as hijacking accounts by abusing password resets.



Utilizing public key cryptography, authentication is performed based on possession using an authenticator.

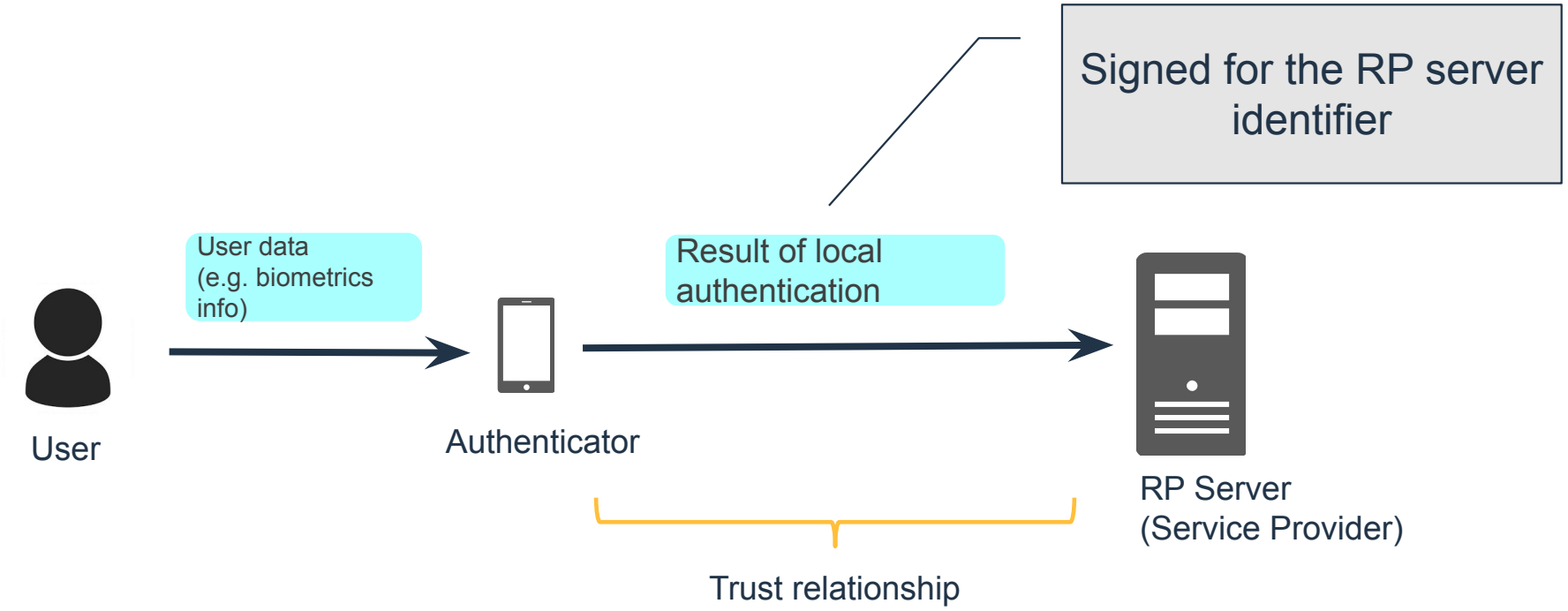


WHAT IS A PASSKEY?

Any passwordless FIDO credential is a passkey.

https://fidoalliance.org/passkeys/

As an example of phishing resistance, the authenticator send the identifier of the RP server with own sign, and the RP server verifies this signature.
= If there is an intermediary, it can be detected by signature verification & Authentication cannot be performed on a server whose domain is different from the RP server.



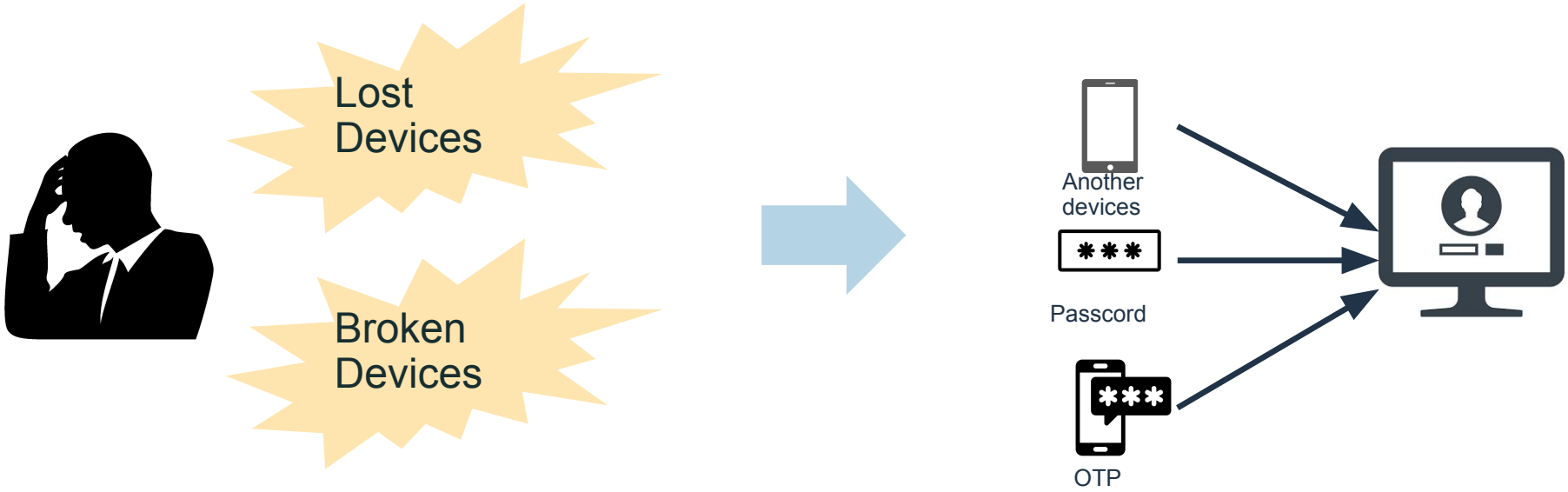
1 History and issues of authentication methods

2 **The release of synced passkeys and related technologies**

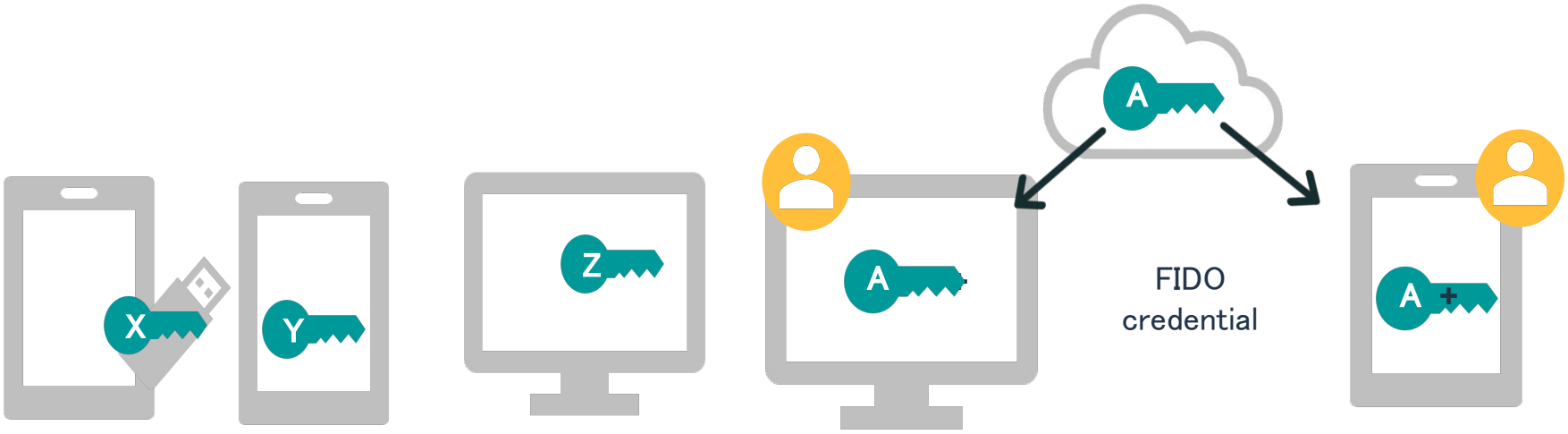
3 Passkey implementation with Ruby

Issues of traditional passkeys

Since the authentication method relies on the authenticator, the problem was how to recover the account if the authenticator itself was lost or damaged.



Passkeys to be saved in a passkey provider (cloud provider or password manager provider) and settings related to FIDO authentication to be migrated as well.
*The need for device-bound credentials continues to be discussed.

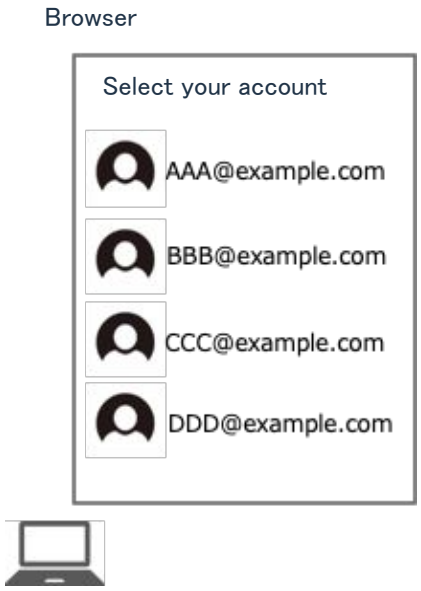


Device-bound passkeys
(Store different key in each devices)

Synced passkeys
(Store same key in each devices)

Check if the current user has credentials and display them as a list. (Passkeys are also included in this list)

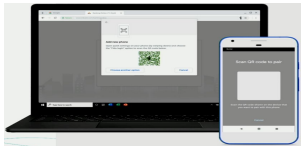
Users no longer need to enter not only a password but also an user id.



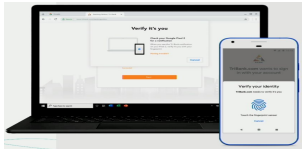
A smartphone can be used as an authenticator. A technology based on Bluetooth that can also be used to leverage Passkeys across platforms.

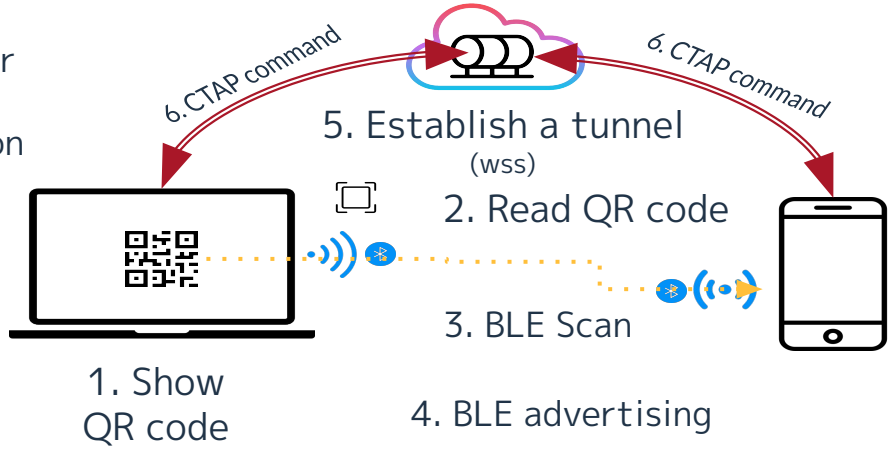
Sign in on your desktop PC

Read the authentication QR code with your smartphone



Verify your identity with your smartphone and the authentication is successful.





Please see a demonstration

Demonstration

https://www.youtube.com/watch?v=_QwOXyoetD8

Comparison with traditional authentication methods -1

Traditional authentication methods and passkeys have different risk scenarios (location/target to protect, entity responsible, impact, etc.)

	Password / One Time Password	Passkeys
Credentials Confidentiality	Password reuse	Generate challenge
	Credential compromise(phishing)	Origin bound keys
	Credential compromise at RP	Loss of public key is not impactful
	Credential compromise (credential manager)	Credential compromise (passkey provider)

Refer authenticate2023
Dean H.Saxe [The New Passkey Threat Model]

Comparison with traditional authentication methods -2

Risk assessment of passkey providers is also important from an availability perspective.
*Passkey provider information can be obtained from AAGUID
https://github.com/passkeydeveloper/passkey-authenticator-aaguids/blob/main/aaguid.json

	Password / One Time Password	Passkeys
Credential Availability	Forgotten	N/A
	Deleted from credential manager	Deleted from authenticator or passkey provider
	Cannot access email/SMS	N/A
	Lost access to credential manager	Device bound passkeys -Lost or broken device
		Synced passkeys -Lost access to passkey provider -forgot credentials / failed account recovery -deleted passkey provider account -cancellation of service

Refer authenticate2023
Dean H.Saxe [The New Passkey Threat Model]をもとに加工

Comparison with traditional authentication methods -3

Correct use of passkeys, such as sharing passkeys using AirDrop, and risk assessment is also important.

	Password / One Time Password	Passkeys
Credential recovery	Account recovery via RP	Device bound passkeys -another device bound credentials -Account recovery via RP
		Synced passkeys -Credential recovery via passkey provider -Account recovery via RP
Credential sharing	Shared via analog mechanisms or voice	N/A
	Shared by value	Apple Airdrop

Refer authenticate2023
Dean H.Saxe [The New Passkey Threat Model]をもとに加工

Support status

MEDLEY

Capability	Android	Chrome OS	iOS/iPad OS	macOS	Ubuntu	Windows
Synced Passkeys	✓ v9+	Planned [1]	✓ v16+	✓ v13+ [2]	✗ Not Supported	Planned [1]
Browser Autofill UI	✓ Chrome Edge ✗ Firefox	Planned	✓ Safari Chrome Edge Firefox	✓ Safari Chrome [2] Edge ✗ Firefox	✗ Not Supported	✓ Chrome [3] Edge Firefox
Cross-Device Authentication ***Authenticator***	✓ v9+	✗ Not Supported	✓ v16+	✗ Not Supported	✗ Not Supported	✗ Not Supported
Cross-Device Authentication ***Client***	Planned	✓ v108+	✓ v16+	✓ v13+	✓ Chrome Edge	✓ v23H2+
Third-Party Passkey Providers	✓ v14+	✗ Not Supported	✓ v17+	✓ v14+	✗ Not Supported	Planned

Advanced Capabilities

Capability	Android	Chrome OS	iOS/iPad OS	macOS	Ubuntu	Windows
***Device-bound* Passkeys**	✗ Not Supported	✗ Not Supported	on security keys	on security keys	on security keys	✓
Device-bound Passkey Attestation	n/a	n/a	n/a	n/a	n/a	✓
Synced Passkey Attestation	✗ Not Supported	n/a	✗ Not Supported	✗ Not Supported	n/a	n/a

https://passkeys.dev/device-support/（2023/10/20）

1 History and issues of authentication methods

2 The release of synced passkeys and related technologies

3 Passkey implementation with Ruby

Passkeys can be implemented using WebAuthn API (Web Authentication API).

The WebAuthn specifications are published as Level 3 (working draft).

https://www.w3.org/TR/webauthn-3/

Multiple Ruby libraries have been released

*Recommend using products that have passed the FIDO Alliance test and are certified.

- webauthn-ruby (Cedarcode)
- devise-passkeys (Ruby Passkeys, wrapper around `webauthn-ruby`)
- warden-webauthn (Ruby Passkeys, wrapper around `webauthn-ruby`)

Sample code using above libraries
webauthn-rails-demo-app

※Other language based libraries also have been released.
awesome-webauthn

Web Authentication API has two basic functions: registration and login.

	Method	Use case	Function
Registration	navigator.credentials.create()	• Passkey registration when registering a user (account) to the service • Passkey registration by existing user (account) of the service	Send signature and public key information
Authentication	navigator.credentials.get()	• Passkey authentication when logging in to existing services • Passkey authentication required for logged in users (accounts)	Send signature

- The registration step requires two APIs to be prepared.

  (API1) API that issues a challenge response linked to a user ID
  (API2) API to verify the public key issued by the authenticator and save it in the database

- Registration step overview

  0. Passkey registration request by user

  1. PublicKeyCredentialCreationOptions object creation by RP (server) (API1)
  2. Execute navigator.credentials.create() by RP(JS APP)
  3. Creation of key pair and attestation signature by authenticator
  4. Return PublicKeyCredential by authenticator
  5. Return AuthenticatorAttestationResponse by RP (JS APP)
  6. Authentication information verified by RP (server), registration completed(API2)

Add library name tto the gemfile

```
gem 'webauthn'
```

Execute $ bundle

```
$ bundle
```

Or execute installation

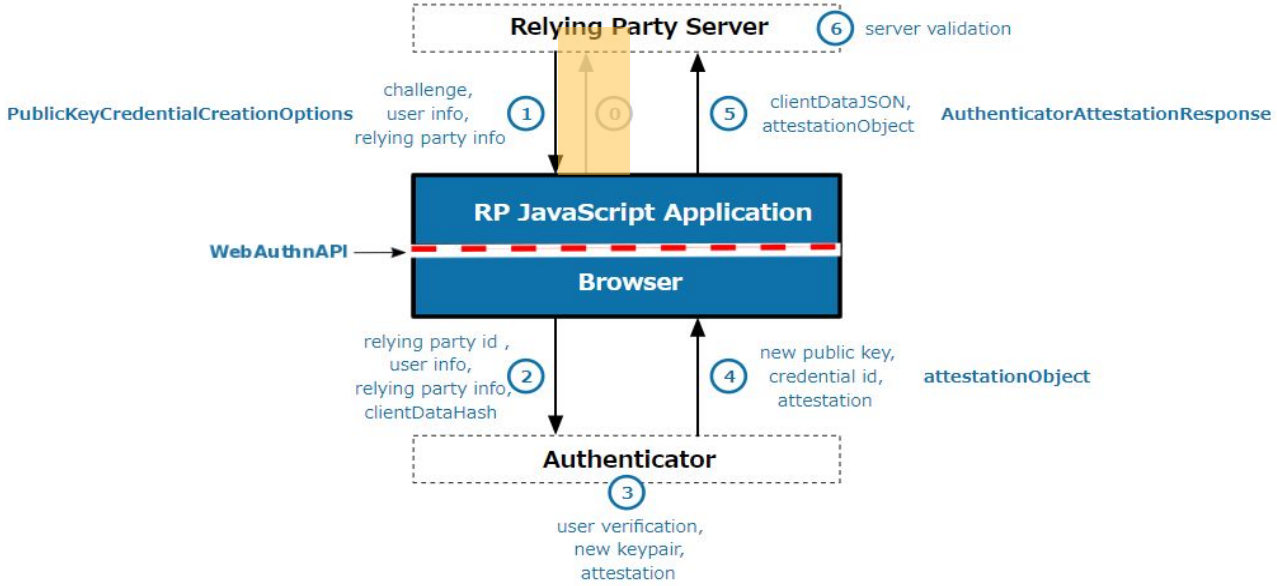
```
$ gem install webauthn
```

Set domain name and RP name etc as config info

```ruby
WebAuthn.configure do |config|
  # This value needs to match `window.location.origin` evaluated by
  # the User Agent during registration and authentication ceremonies.
  config.origin = "https://auth.example.com"

  # Relying Party name for display purposes
  config.rp_name = "Example Inc."
end
```

0. Passkey registration request by user

The browser display a button such as "Register Passkey" and user sends the information necessary for account registration to the server through events such as pressing the button. The information required for registration differs depending on the service. (name, address, etc.)

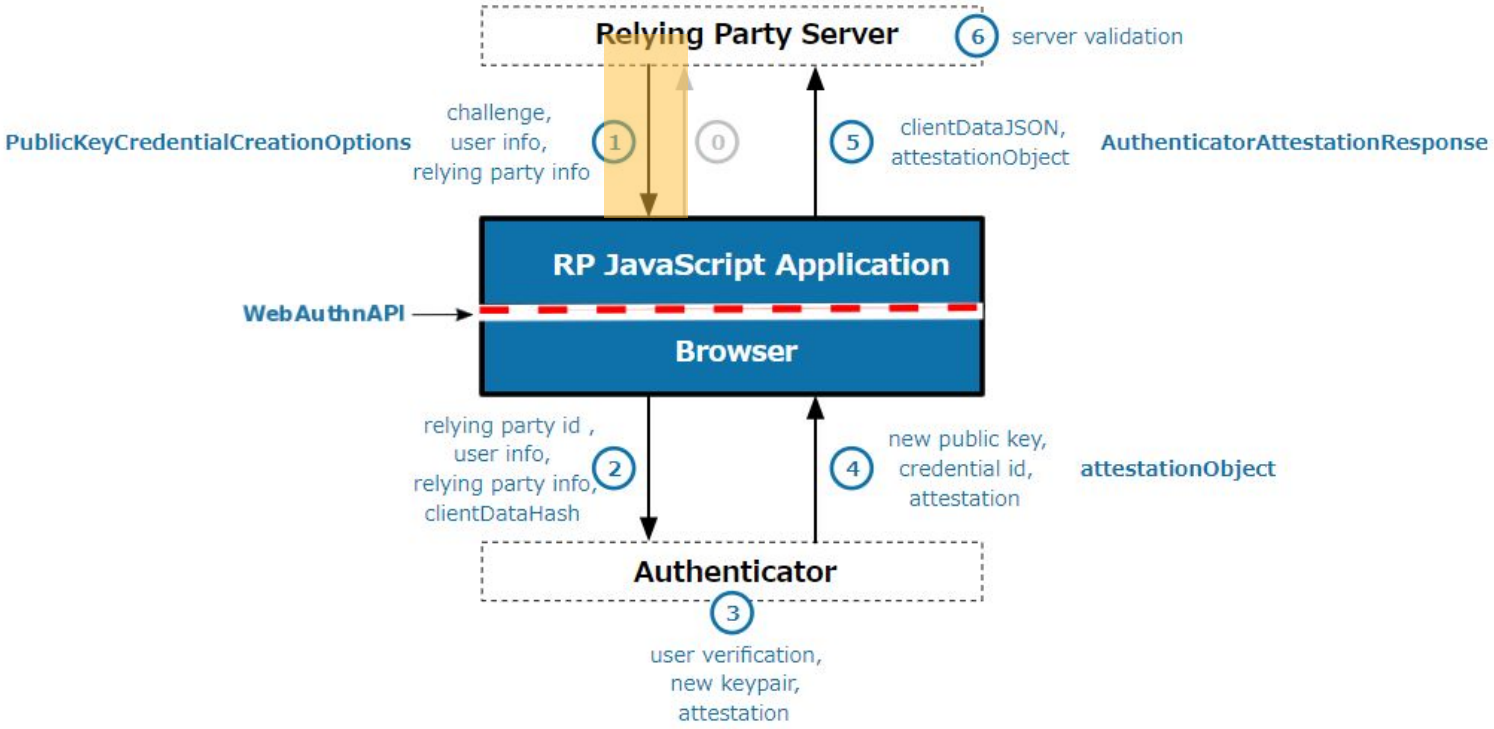



```
<%= form_with url: registration_path, .... do |form| %>
    <%= form.text_field :username %>
    <%= form.submit 'Registration' %>
<% end %>
```

1. PublicKeyCredentialCreationOptions object creation by RP (server)

MEDLEY

When RP received request from a user, RP generates a challenge (random number) and creates a PublicKeyCredentialCreationOptions object along with information necessary for the next step such as RP information.



1. PublicKeyCredentialCreationOptions object creation by RP (server)

MEDLEY

Excerpt from contents of PublicKeyCredentialCreationOptions Object

Parameter	Must	Description
rp	✓	Relying Party info id:identification（domain name） name: RP name
user	✓	User info id: identification in RP name: user name displayName:display name
challenge	✓	Random number generated by server
pubKeyCredParams	✓	Key info type: Credential type alg: encryption algorithm
AuthenticatorSelection		Available function on authenticator AuthenticatorAttachement :control syncronization to platform requireResidentkey : Discoverable Credential usage userVerification : requirement level for local authentication
timeout		Waiting time for API calling
excludeCredentials		Existing passkey info (to avoid register same passkey)

1. PublicKeyCredentialCreationOptions object creation by RP (server)

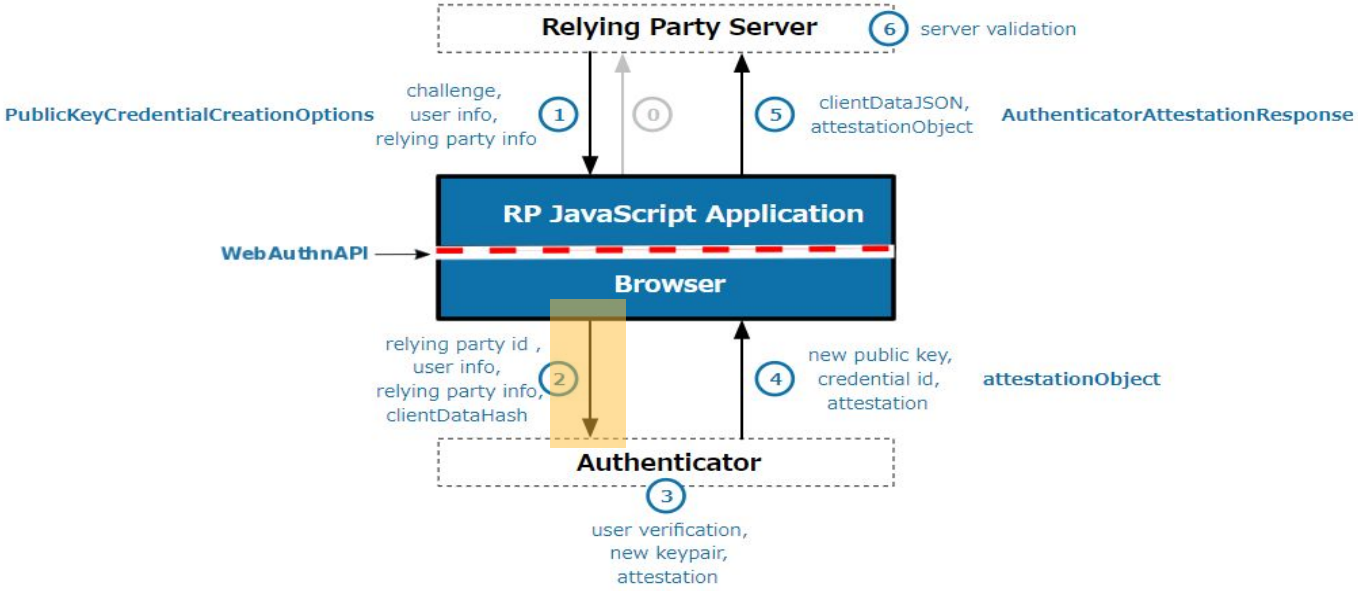
MEDLEY

```ruby
def create
  user = User.new(username: params[:registration][:username])

  create_options = WebAuthn::Credential.options_for_registration(
    user: {
      name: params[:registration][:username],
      id: user.webauthn_id
    },
    authenticator_selection: {
        user_verification: "required"
        require_resident_key:true
        authenticatorAttachment: "platform" }
  )

  if user.valid?
    session[:current_registration] = { challenge: create_options.challenge, user_attributes: user.attributes }

    respond_to do |format|
      format.json { render json: create_options }
    end
  else
    respond_to do |format|
      format.json { render json: { errors: user.errors.full_messages }, status: :unprocessable_entity }
    end
  end
end
```

2. Execute navigator.credentials.create() by RP(JS APP)

MEDLEY

Call the navigator.credentials.create() method with the PublicKeyCredentialCreationOptions object and PublicKey as arguments




```
const cred = await navigator.credentials.create({
  publicKey: options,
});
```

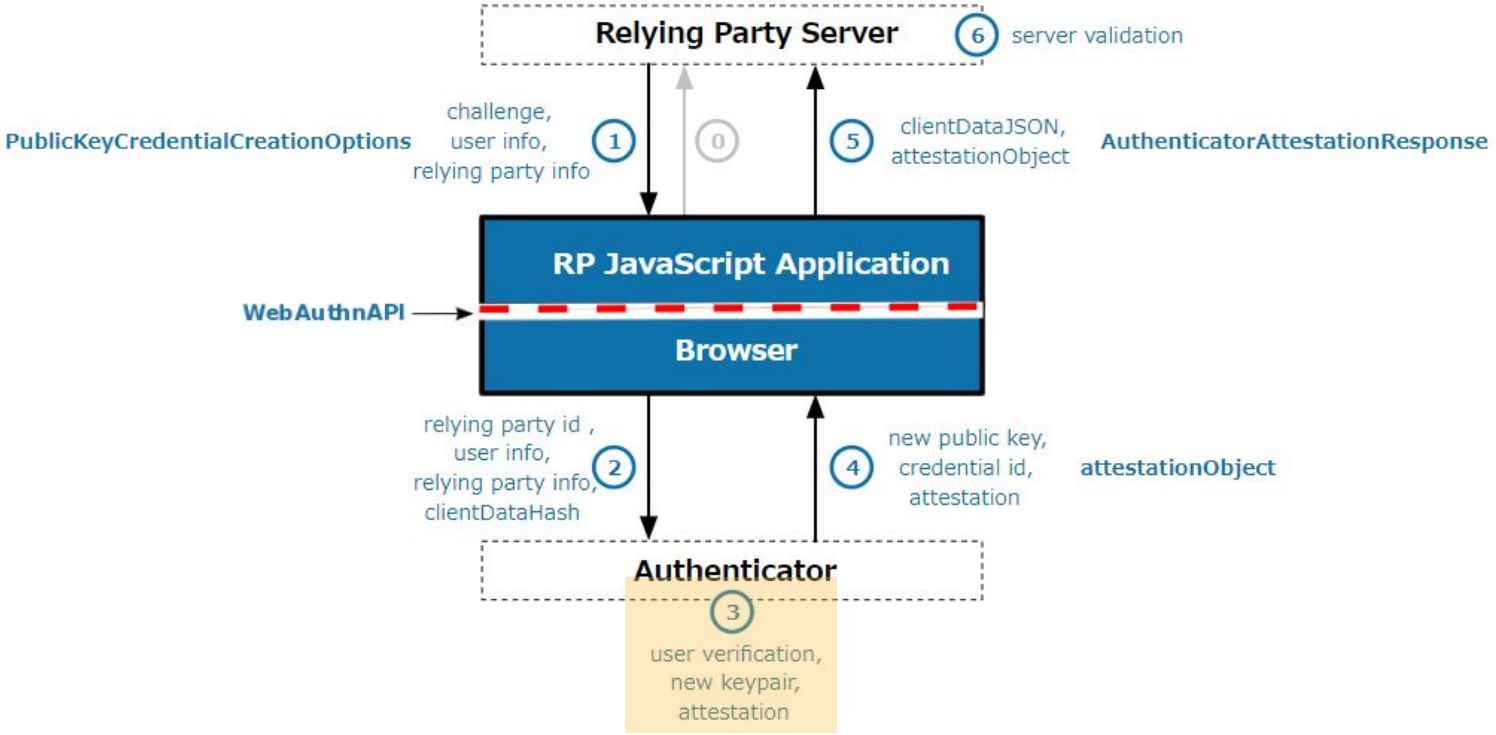
There are several JavaScript library

例）[webauthn-json]

3. Creation of key pair and attestation signature by authenticator

MEDLEY

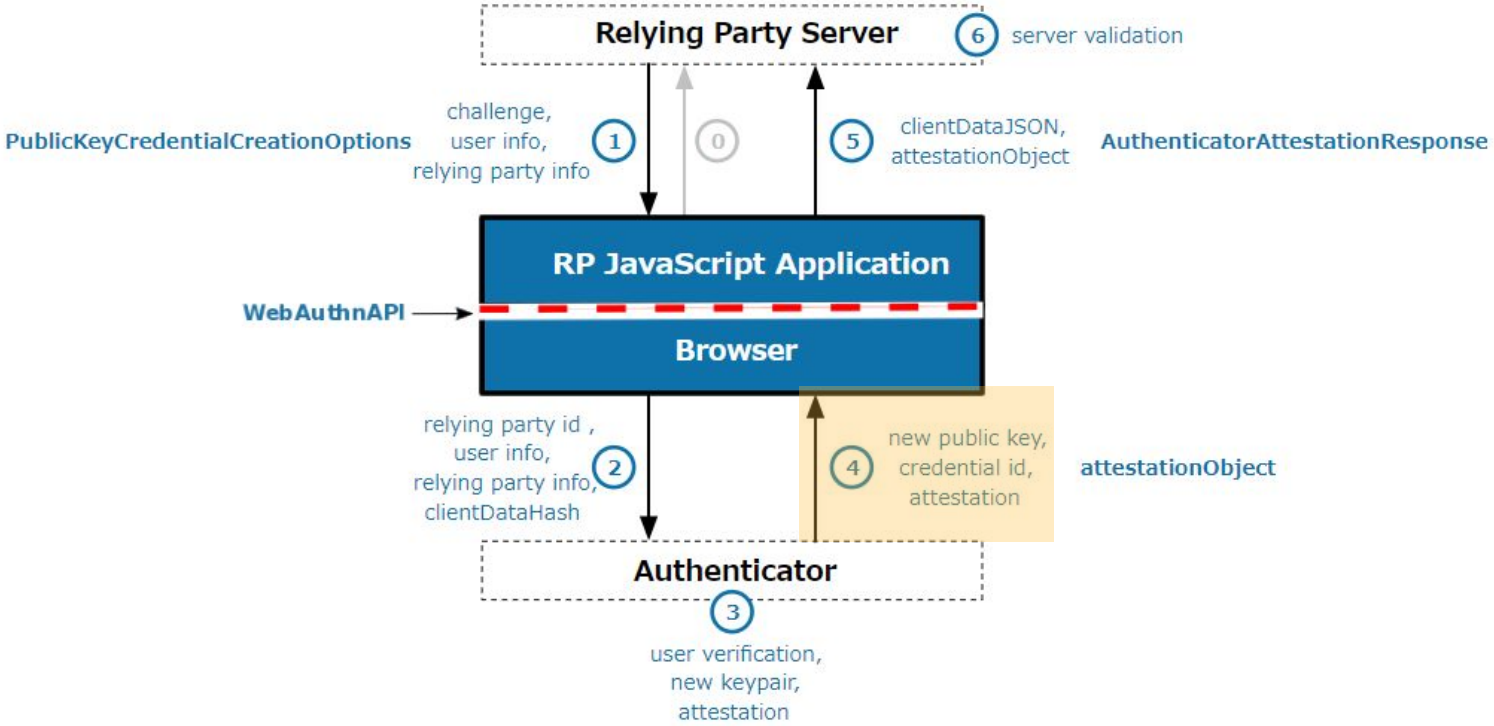
When local authentication (identity verification) with the authenticator is successful, create a key pair and sign it with the private key for attestation.



4. Return PublicKeyCredential by authenticator

MEDLEY

The authenticator returns the created public key and authentication information such as Attestation to the browser as an attestationObject.



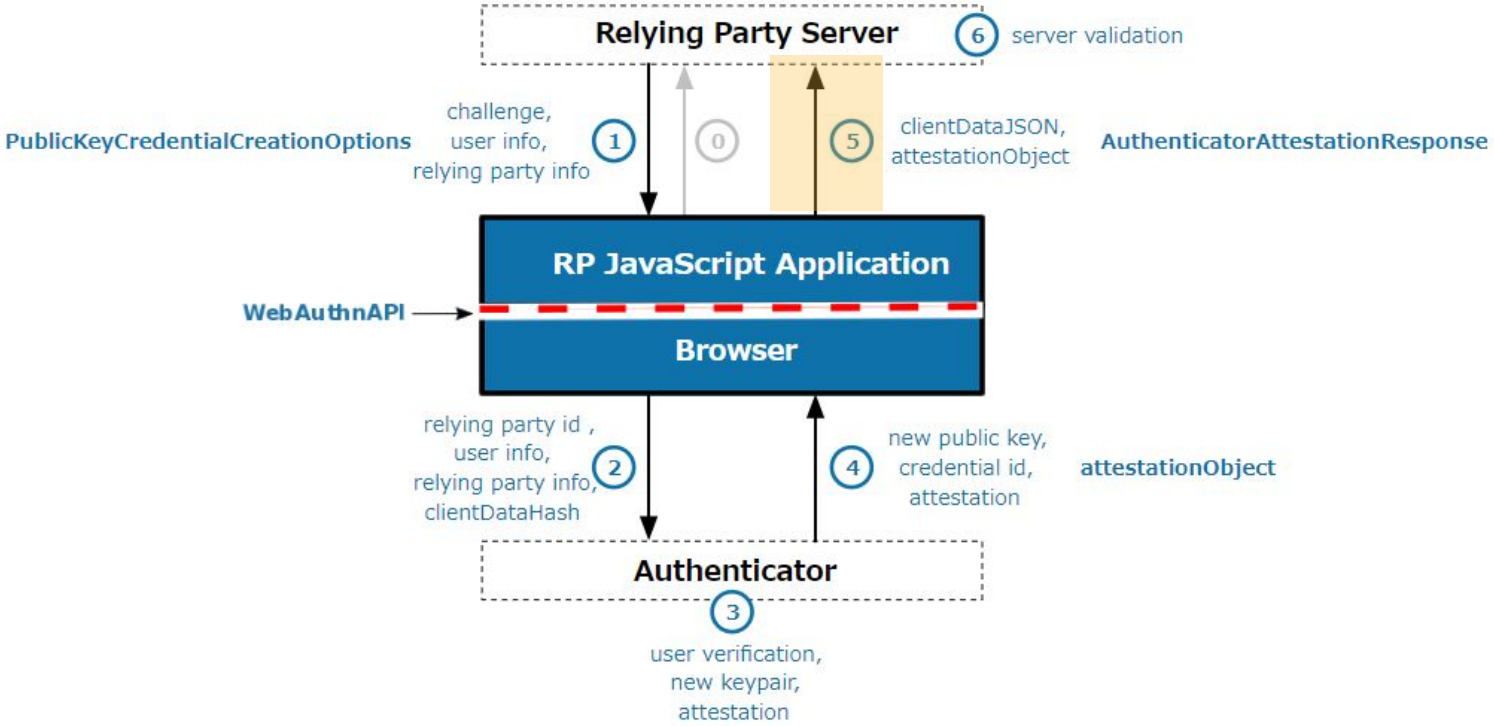
Excerpt from contents of PublicKeyCredential

Parameter	Description
rawId	Binary data of generated public key identifier (ArrayBuffer)
id	The generated public key identifier is actually the Base64url encoded data of the rawId above
response	Authenticator response to credential generation request (AuthenticatorAttestationResponse) attestationObject As a result of user verification by the authenticator, authenticator data and attestation information in COSE format clientDataJSON JSON format information of the information (ClientData) sent from the web browser to the authenticator when requesting credential generation
type	PublicKeyCredential type。generally, public-key

5.Return AuthenticatorAttestationResponse by RP (JS APP)

MEDLEY

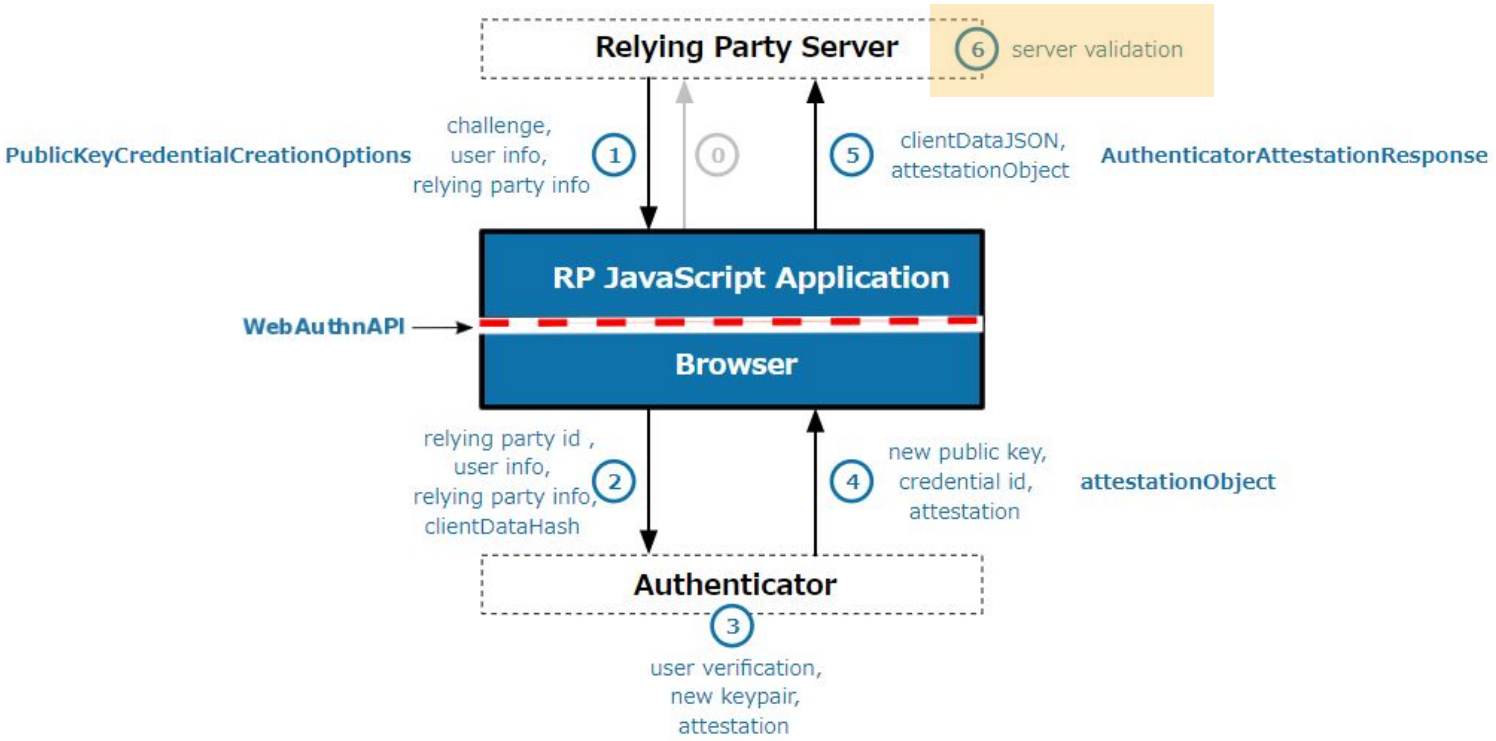
Send authentication info to AttesstationResponse endpoint at RP



6.Authentication information verified by RP (server), registration completed

RP verify received authentication info. After successfully verified, return appropriate success response and store public key to DB for using authentication

Registration has been completed.



6.Authentication information verified by RP (server), registration completed

MEDLEY

```ruby
def callback
  webauthn_credential = WebAuthn::Credential.from_create(params)

  begin
    webauthn_credential.verify(
      session["current_registration"]["challenge"],
      user_verification: true
      )

    credential = user.credentials.build(
      external_id: Base64.strict_encode64(webauthn_credential.raw_id),
      nickname: params[:credential_nickname],
      public_key: webauthn_credential.public_key,
      sign_count: webauthn_credential.sign_count
    )

    if credential.save
      sign_in(user)

      render json: { status: "ok" }, status: :ok
    else
      render json: "Couldn't register your Security Key", status: :unprocessable_entity
    end
  rescue WebAuthn::Error => e
    render json: "Verification failed: #{e.message}", status: :unprocessable_entity
  ensure
    session.delete("current_registration")
  end
end
```

- The authentication step requires two APIs to be prepared.

  （API1）API that issues a challenge response linked to a user ID
  （API2）API that issues a session by verifying the signature issued by the browser's authenticator using a public key stored on the server.

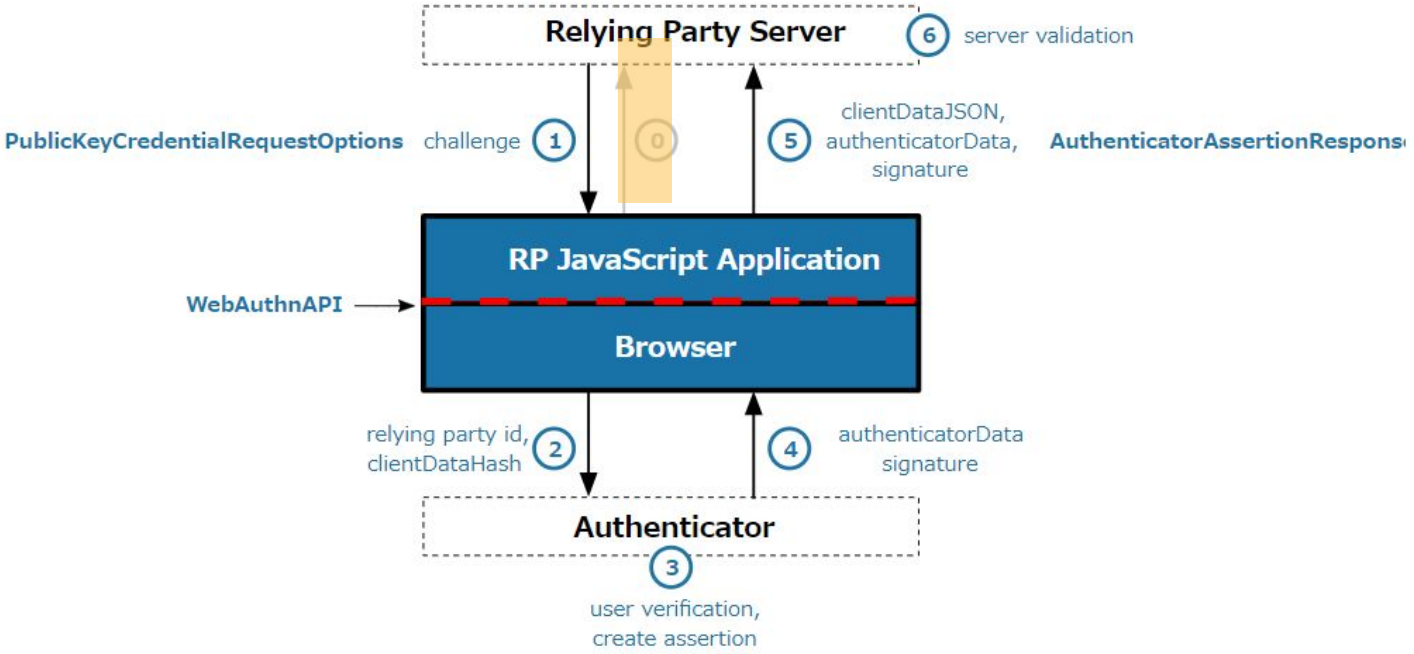
- Authentication step overview

  0. Passkey authentication request by user
  1. Create PublicKeyCredentialRequestOptions Object by RP（Server）（API1）
  2. Execute navigator.credentials.get() by RP（JS APP）
  3. User confirmation and assertion creation by authenticator
  4. Return authentication info by authenticator
  5. Return AuthenticationAssertionResponse by RP（JS APP）
  6. Signature verification and authentication completed by RP (server)(API2)

0 Passkey authentication request by user

MEDLEY

Similar to registration, request specifications for authentication endpoints are also outside the scope of WebAuthn specifications. Usually uses a unique identifier such as a user ID that can identify the user.




```
<%= form_with url: session_path, .... do |form| %>
    <%= form.text_field :username %>
    <%= form.submit 'Login' %>
<% end %>
```

0 Passkey authentication request by user

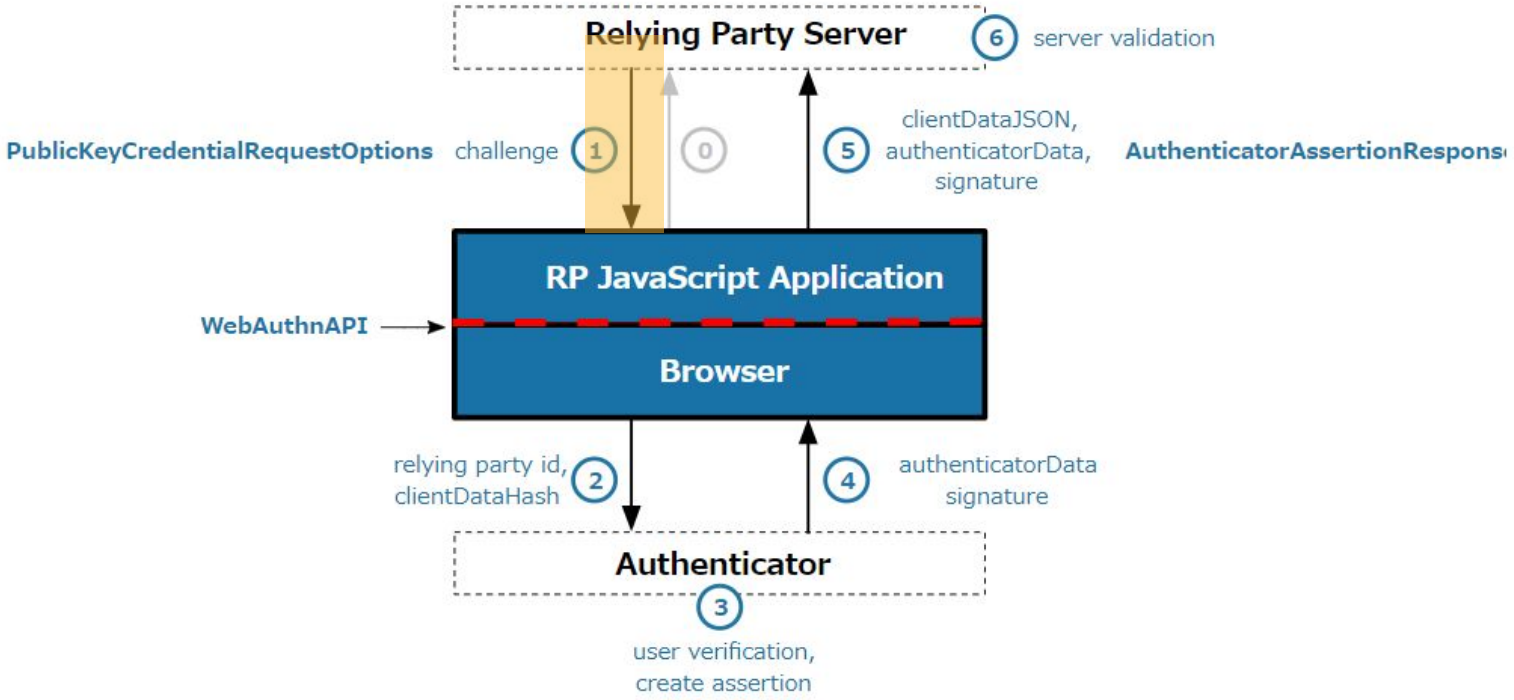
MEDLEY

If you want to use Autofill, specify the autocomplete attribute in the input field of the login page.

```
<input
    required
    type="email"
    name="email"
    autoComplete="username webauthn"
/>
```

1. Create PublicKeyCredentialRequestOptions Object by RP (Server)

Upon receiving a request from a user, the server generates a challenge (random number). Also, store the public key information created by the authenticator stored in the DB in allowCredentials and create a PublicKeyCredentialRequestOptions object.



1. Create PublicKeyCredentialRequestOptions Object by RP（Server）

```ruby
def create
  user = User.find_by(username: session_params[:username])

  if user
    get_options = WebAuthn::Credential.options_for_authentication(
      allow: user.credentials.pluck(:external_id),
      user_verification: "required"
    )

    session[:current_authentication] = { challenge: get_options.challenge, username: session_params[:username] }

    respond_to do |format|
      format.json { render json: get_options }
    end
  else
    respond_to do |format|
      format.json { render json: { errors: ["Username doesn't exist"] }, status: :unprocessable_entity }
    end
  end
end
```

1. Create PublicKeyCredentialRequestOptions Object by RP（Server）

MEDLEY

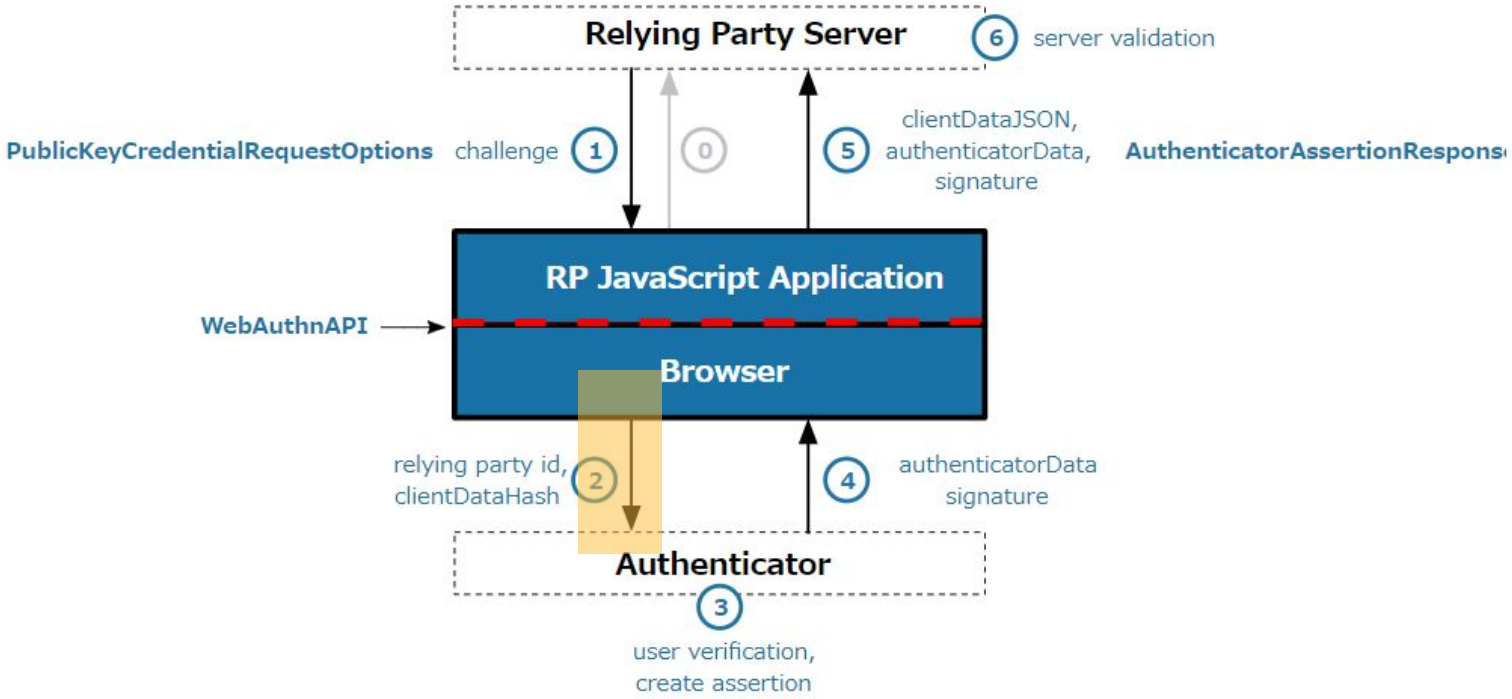
Excerpt from contents of PublicKeyCredentialRequestOptions Object

Parameter	Must	Descriotion
challenge	✓	Random number generated by server
allowCredentials		type: PublicKeyCredential type id: CredentialID by executing create() transports: Communication method to authenticator

2. Execute navigator.credentials.get() by RP（JS APP）

MEDLEY

Based on the RP response, execute navigator.credentials.get() to perform authentication processing.



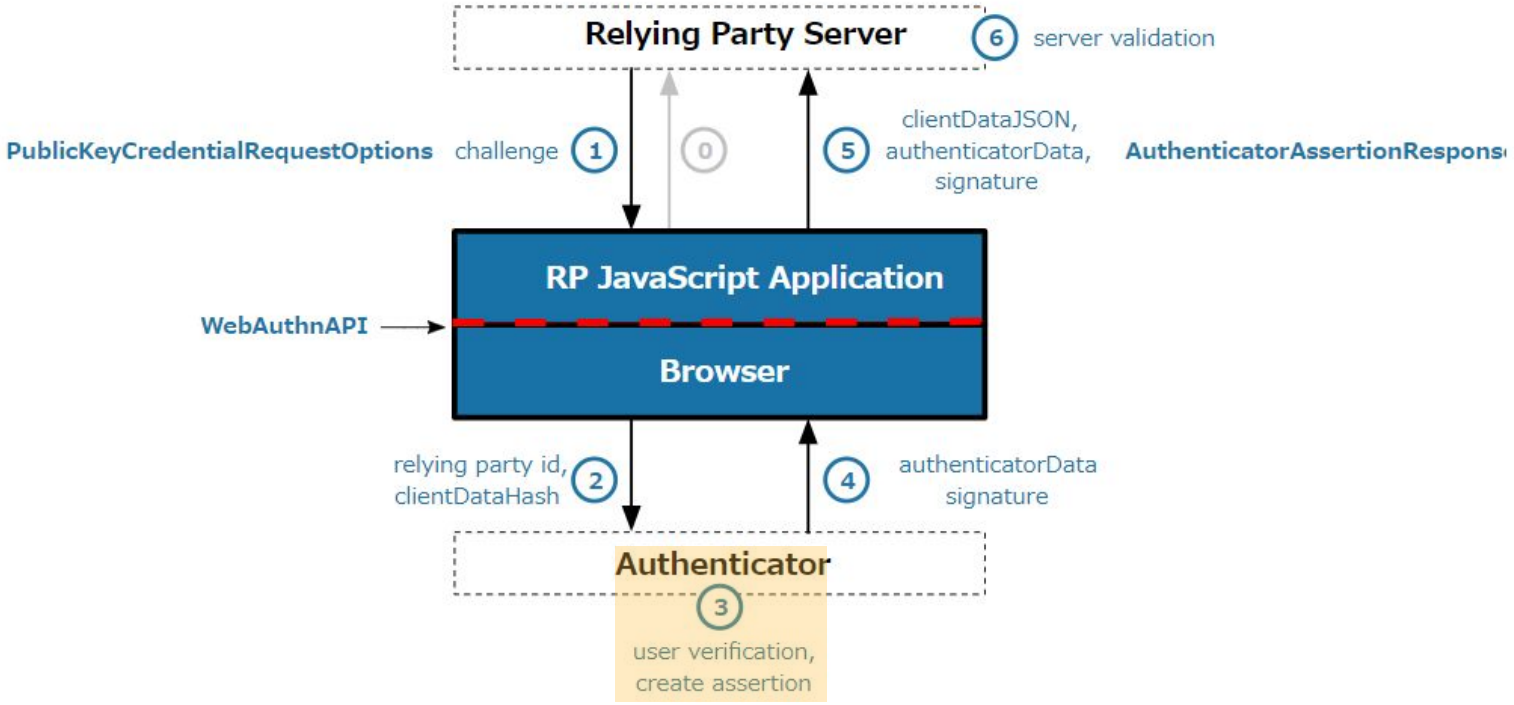
2. Execute navigator.credentials.get() by RP（JS APP）

MEDLEY

Pass the PublicKeyCredentialRequestOptions object with publickKey as an argument.

```
const cred = await navigator.credentials.get({
  publicKey: options,
  // Request a conditional UI
  mediation: conditional ? 'conditional' : 'optional'
});
```

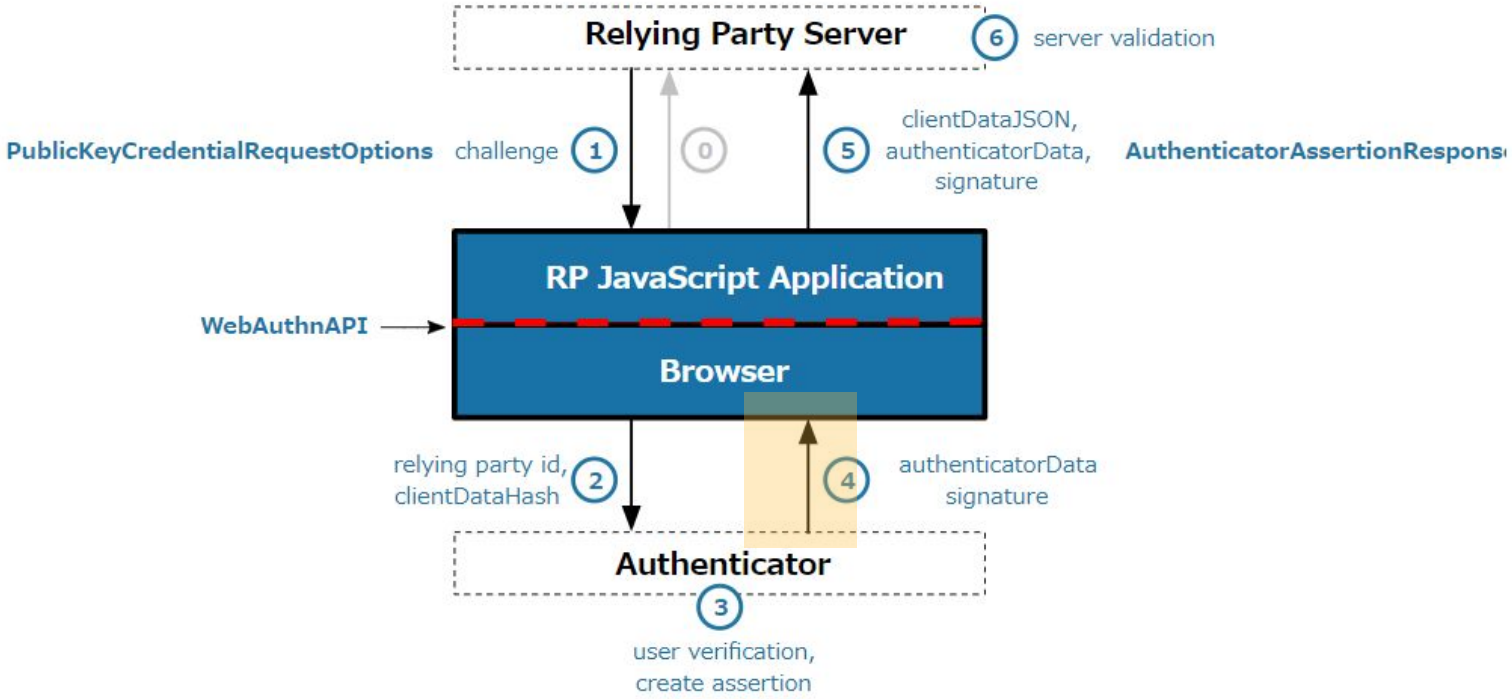
The authenticator verifies the user based on the Challenge and RelyingParty information and creates an assertion.



4. Return PublicKeyCredential by authenticator

MEDLEY

Returns the PublicKeyCredential (assertion signature, clientDataHash, AuthenticatorData, etc.) generated by the authenticator to the browser.

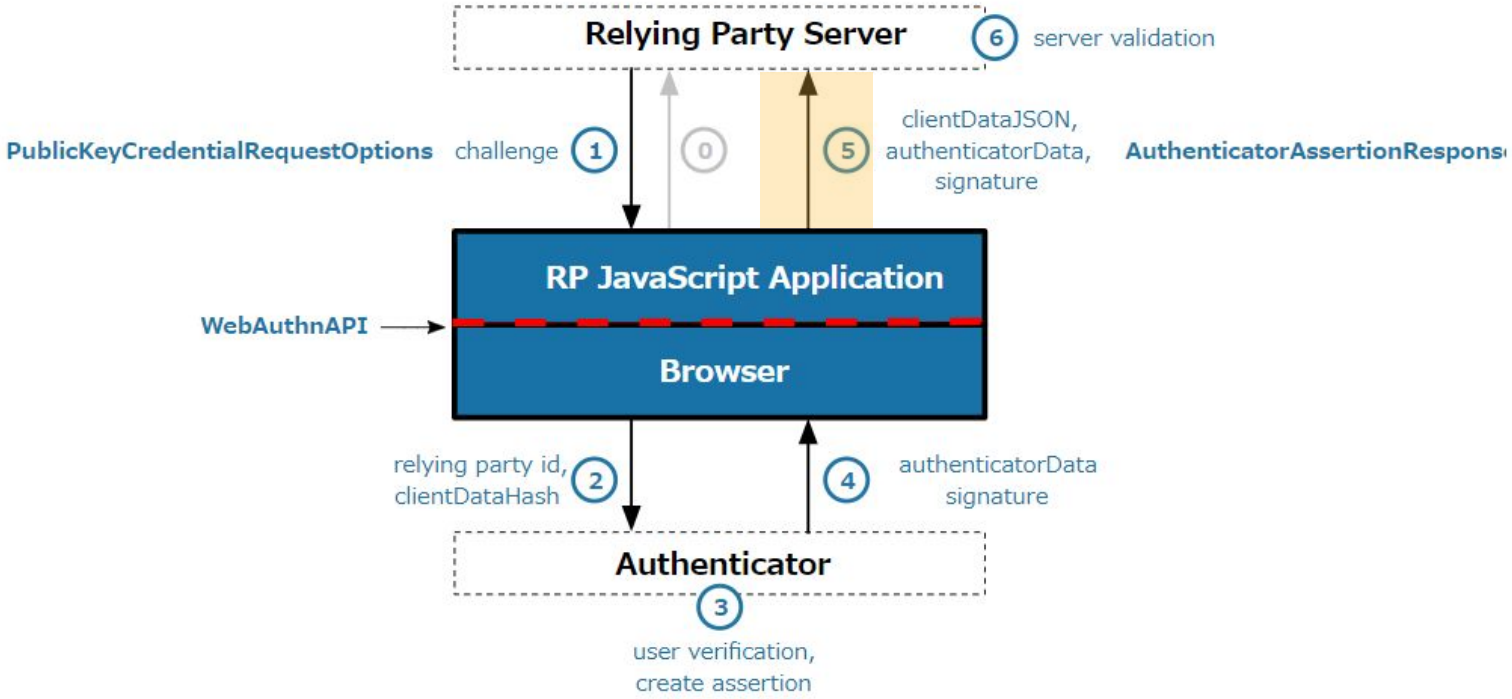


Excerpt from contents of PublicKeyCredential

Parameter	Description
rawId	Binary data of public key identifier（ArrayBuffer）
id	The public key identifier is actually the Base64url encoded data of the rawId above.
response	Authenticator response to credential authentication request (AuthenticatorAssertionResponse) AuthenticatorData Metadata about the authenticator clientDataJSON JSON format information of the information (ClientData) sent from the web browser to the authenticator when requesting credential authentication signature Signature data for PublicKeyCredential generated by the authenticator. It is generated using the registered user's private key for data that is a concatenation of the hash values (ClienthDataHash) of the above authenticatorData and ClientData.
type	PublicKeyCredential type. Generally, public-key

5.Return AuthenticationAssertionResponse by RP（JS APP）

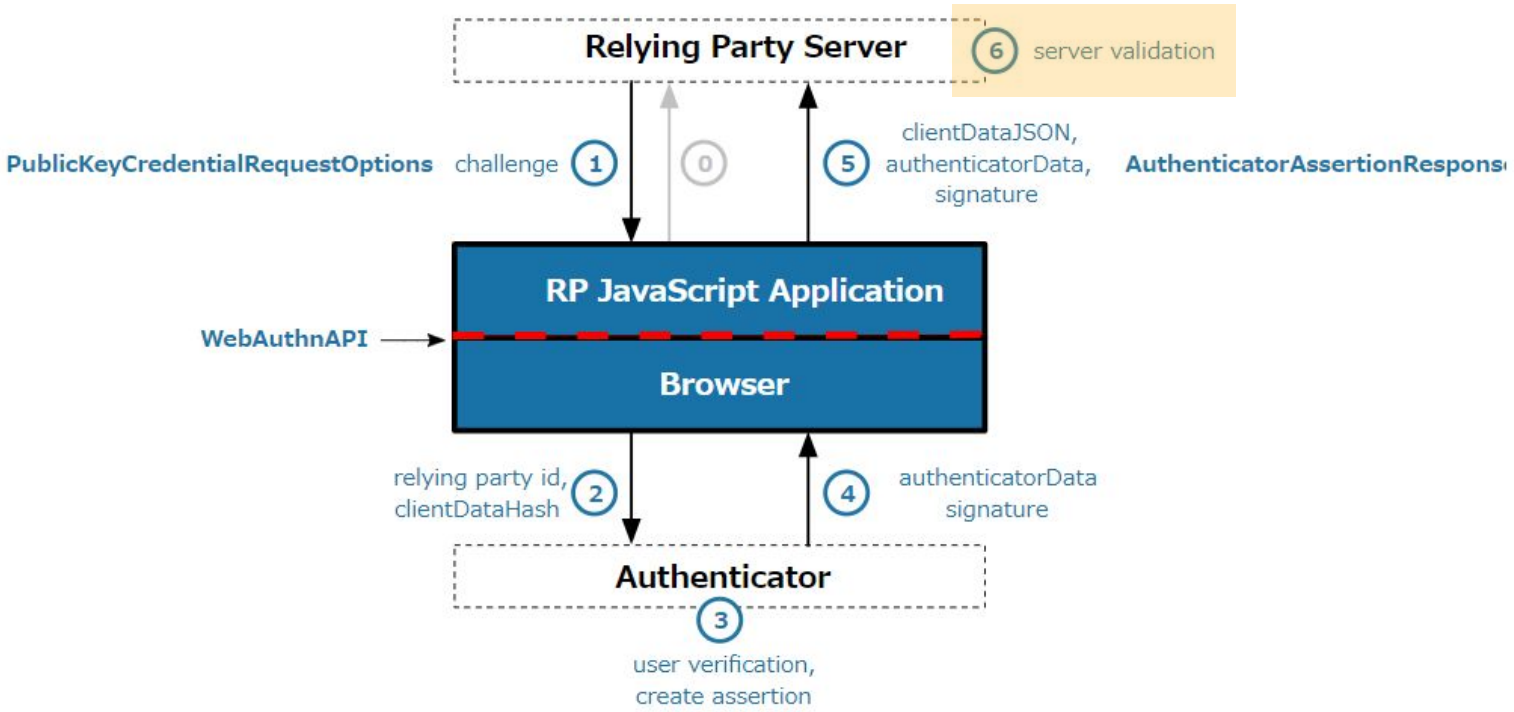
As with registration, in addition to the parameters received from the authenticator, include parameters such as rawId, id, type, clienetDataJSON, etc., and send to the AssertionResponse endpoint.



6.Signature verification and authentication completed by RP (server)

MEDLEY

The RP verifies the Assertion, and if the verification is successful, the authentication to be successful. Continue processing in the same way as a general authentication server, such as issuing a session and setting the session ID in a cookie.



6.Signature verification and authentication completed by RP (server)

MEDLEY

```ruby
def callback
  user = User.find_by(username: session["current_authentication"]["username"])
  raise "user #{session["current_authentication"]["username"]} never initiated sign up" unless user

  begin
    verified_webauthn_credential, stored_credential = relying_party.verify_authentication(
      params,
      session["current_authentication"]["challenge"],
      user_verification: true,
    ) do |webauthn_credential|
      user.credentials.find_by(external_id: Base64.strict_encode64(webauthn_credential.raw_id))
    end

    stored_credential.update!(sign_count: verified_webauthn_credential.sign_count)
    sign_in(user)

    render json: { status: "ok" }, status: :ok
  rescue WebAuthn::Error => e
    render json: "Verification failed: #{e.message}", status: :unprocessable_entity
  ensure
    session.delete("current_authentication")
  end
end
```

- Passkeys that are secure and improve usability are now even easier to adapt products (Ruby libraries and sample code are also available)

- Implementation with attention to precautions unique to Synced Passkey and ongoing issues such as;
  - Security level of Passkey providers
  - Confirming device bound credentials

- Pay attention to smooth transition from existing authentication UX
  - UX guidelines have also been published.