Latest Updates to the Credential Management API

By Eiji Kitamura
Developer Advocate in Tokyo

Some of the updates described here are explained in the Google I/O session, **Secure and Seamless Sign-In: Keeping Users Engaged**:

Chrome 57

Chrome 57 introduced this important change to the Credential Management API.

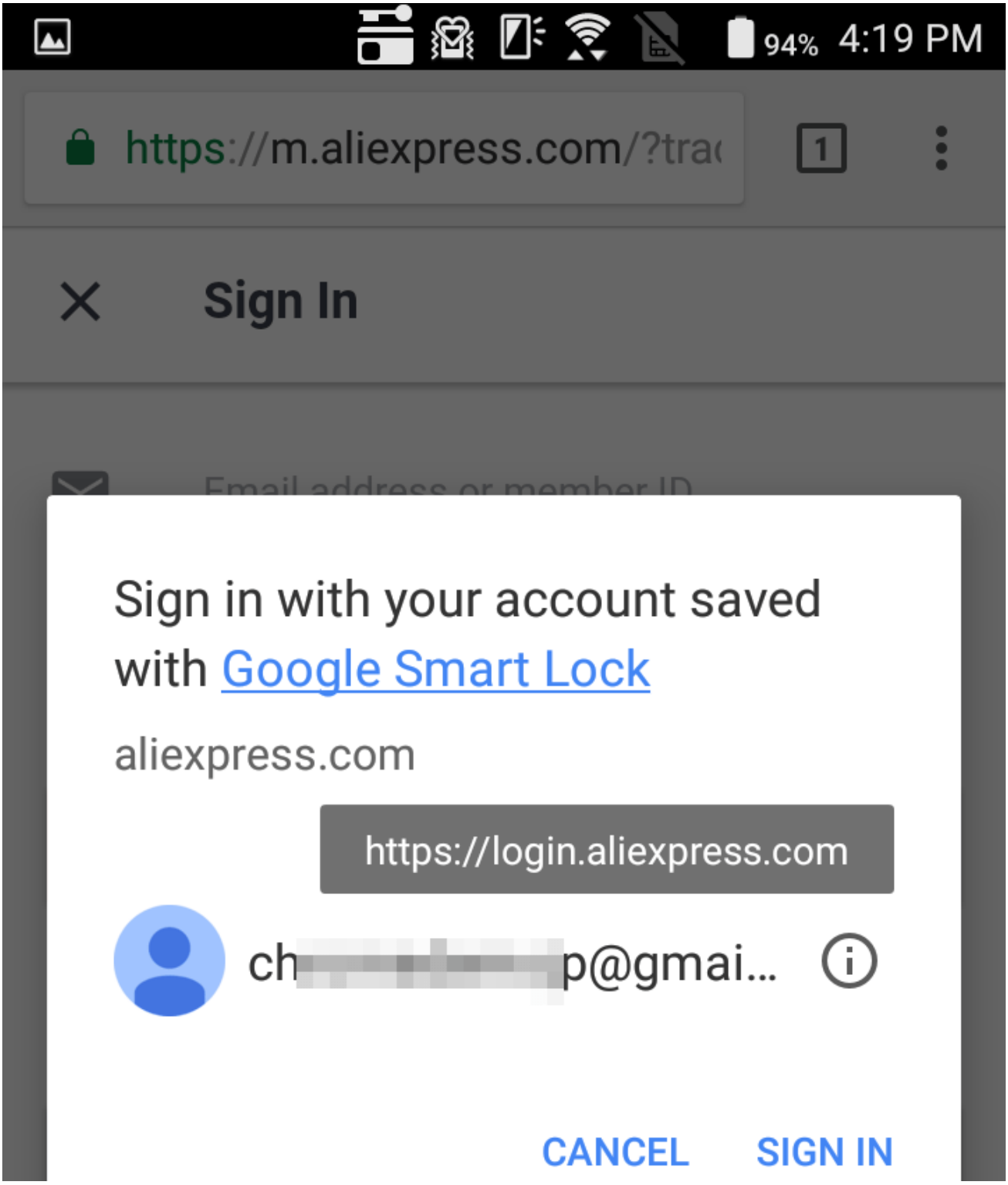
Credentials can be shared from a different subdomain

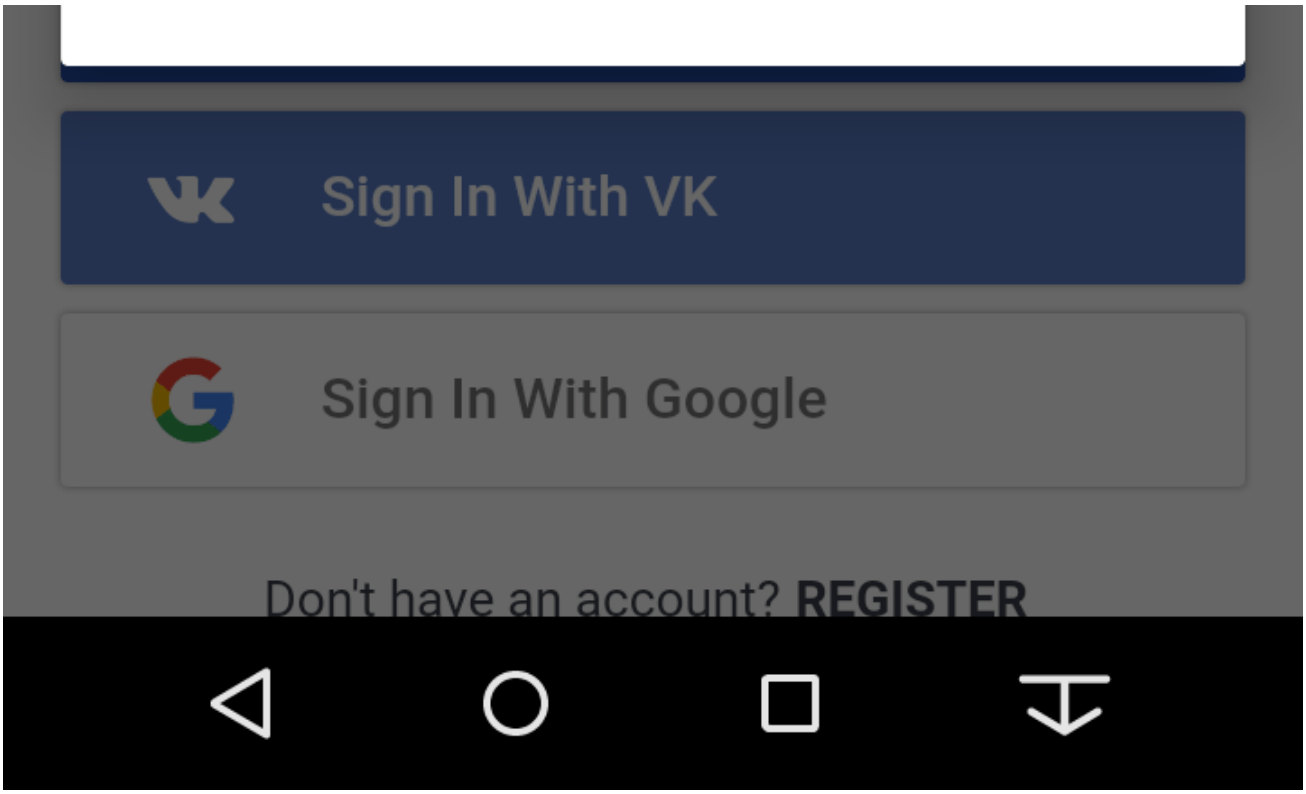
Chrome can now retrieve a credential stored in a different subdomain using the Credential Management API. For example, if a password is stored in `login.example.com`, a script on `www.example.com` can show it as one of account items in account chooser dialog.

You must explicitly store the password using `navigator.credentials.store()`, so that when a user chooses a credential by tapping on the dialog, the password gets passed and copied to the current origin.

Once it's stored, the password is available as a credential in the exact same origin `www.example.com` onward.

In the following screenshot, credential information stored under `login.aliexpress.com` is visible to `m.aliexpress.com` and available for the user to choose from:





Coming soon: Sharing credentials between totally different domains is in development.

Chrome 60

Chrome 60 introduces several important changes to the Credential Management API:

- `PasswordCredential` object now includes a password.
- As the custom `fetch()` function is no longer required to fetch the password, it will be deprecated soon.
- `navigator.credentials.get()` now accepts an enum `mediation` instead of boolean flag `unmediated`.
- `requireUserMediation()` renamed to `preventSilentAccess()`.
- New method `navigator.credentials.create()` asynchronously creates credential objects.

Feature detection needs attention

To see if the Credential Management API for accessing password-based and federated credentials is available, check if `window.PasswordCredential` or

`window.FederatedCredential` is available.

```
if (window.PasswordCredential || window.FederatedCredential) {
  // The Credential Management API is available
}
```

Warning: Feature detection by checking `navigator.credentials` may break your website on browsers supporting WebAuthn(PublicKeyCredential) but not all credential types (`PasswordCredential` and `FederatedCredential`) defined by the Credential Management API. Learn more.

`PasswordCredential` object now includes password

The Credential Management API took a conservative approach to handling passwords. It concealed passwords from JavaScript, requiring developers to send the `PasswordCredential` object directly to their server for validation via an extension to the `fetch()` API.

But this approach introduced a number of restrictions. We received feedback that developers could not use the API because:

- They had to send the password as part of a JSON object.
- They had to send the hash value of the password to their server.

After performing a security analysis and recognizing that concealing passwords from JavaScript did not prevent all attack vectors as effectively as we were hoping, we have decided to make a change.

The Credential Management API now includes a raw password in a returned credential object so you have access to it as plain text. You can use existing methods to deliver credential information to your server:

```
navigator.credentials.get({
  password: true,
  federated: {
    providers: [ 'https://accounts.google.com' ]
  },
  mediation: 'silent'
}).then(passwordCred => {
  if (passwordCred) {
    let form = new FormData();
    form.append('email', passwordCred.id);
    form.append('password', passwordCred.password);
```

```
    form.append('csrf_token', csrf_token);
    return fetch('/signin', {
      method: 'POST',
      credentials: 'include',
      body: form
    });
  } else {

    // Fallback to sign-in form
  }
}).then(res => {
  if (res.status === 200) {
    return res.json();
  } else {
    throw 'Auth failed';
  }
}).then(profile => {
  console.log('Auth succeeded', profile);
});
```

Custom fetch will be deprecated soon

Warning: Now that passwords are returned in the `PasswordCredential` object, the custom `fetch()` function will stop working in Chrome 64 (expected 23 Jan 2018. Developers **must** update their code.

To determine if you are using a custom `fetch()` function, check if it uses a `PasswordCredential` object or a `FederatedCredential` object as a value of `credentials` property, for example:

```
fetch('/signin', {
  method: 'POST',
  credentials: c
})
```

Using a regular `fetch()` function as shown in the previous code example, or using an `XMLHttpRequest` is recommended.

`navigator.credentials.get()` now accepts an enum mediation

Until Chrome 60, `navigator.credentials.get()` accepted an optional `unmediated` property with a boolean flag. For example:

```
navigator.credentials.get({
  password: true,
  federated: {
    providers: [ 'https://accounts.google.com' ]
  },
  unmediated: true
}).then(c => {

  // Sign-in
});
```

Setting **`unmediated: true`** prevents the browser from showing the account chooser when passing a credential.

The flag is now extended as mediation. The user mediation could happen when:

- A user needs to choose an account to sign-in with.
- A user wants to explicitly sign-in after **`navigator.credentials.requireUseMediation()`** call.

Choose one of the following options for the **`mediation`** value:

`mediation` value	Compared to boolean flag	Behavior
`silent`	Equals **`unmediated: true`**	Credential passed without showing an account chooser.
`optional`	Equals **`unmediated: false`**	Shows an account chooser if **`preventSilentAccess()`** called previously.
`required`	A new option	Always show an account chooser. Useful when you want to let a user to switch an account using the native account chooser dialog.

In this example, the credential is passed without showing an account chooser, the equivalent of the previous flag, **`unmediated: true`**:

```
navigator.credentials.get({
  password: true,
  federated: {
    providers: [ 'https://accounts.google.com' ]
  },
  mediation: 'silent'
}).then(c => {
```

```
  // Sign-in
});
```

`requireUserMediation()` renamed to `preventSilentAccess()`

To align nicely with the new **`mediation`** property offered in the **`get()`** call, the **`navigator.credentials.requireUserMediation()`** method has been renamed to **`navigator.credentials.preventSilentAccess()`**.

The renamed method prevents passing a credential without showing the account chooser (sometimes called without user mediation). This is useful when a user signs out of a website or unregisters from one and doesn't want to get signed back in automatically at the next visit.

```
signoutUser();
if (navigator.credentials) {
  navigator.credentials.preventSilentAccess();
}
```

Create credential objects asynchronously with new method `navigator.credentials.create()`

You now have the option to create credential objects asynchronously with the new method, **`navigator.credentials.create()`**. Read on for a comparison between both the sync and async approaches.

Creating a `PasswordCredential` object

Sync approach

```
let c = new PasswordCredential(form);
```

Async approach (new)

```
let c = await navigator.credentials.create({
  password: form
});
```

or:

```
let c = await navigator.credentials.create({
  password: {
    id: id,
    password: password
  }
});
```

Creating a `FederatedCredential` object

Sync approach

```
let c = new FederatedCredential({
  id:       'agektmr',
  name:     'Eiji Kitamura',
  provider: 'https://accounts.google.com',
  iconURL:  'https://*****'
});
```

Async approach (new)

```
let c = await navigator.credentials.create({
  federated: {
    id:       'agektmr',
    name:     'Eiji Kitamura',
    provider: 'https://accounts.google.com',
    iconURL:  'https://*****'
  }
});
```

Migration guide

Have an existing implementation of the Credential Management API? We have a migration guide doc you can follow to upgrade to the new version.

Last updated July 2, 2018.