Upgrade Flags SDK to v4

Follow these steps to upgrade to Flags SDK v4.

Ensure latest version

Ensure you are on version 4 or above of the `flags` package.

We recently renamed the `@vercel/flags` package to `flags`. Move to the new `flags` package if you are still using the `@vercel/flags` package. It offers the same functionality under a new, shorter name. The previous `@vercel/flags` package is deprecated.

Adjust flags discovery endpoint

Your application returns metadata about its feature flags to authenticated clients like the Flags Explorer through the flags discovery endpoint under `/.well-known/vercel/flags`.

Flags Explorer now requires your flags discovery endpoint (`/.well-known/vercel/flags`) to return the version of your Flags SDK package as a `x-flags-sdk-version` response header for all successful responses. It is not required when responding with a 401 or other error codes.

Next.js App Router

If you are using Next.js App Router, we recommend switching to the new `createFlagsDiscoveryEndpoint` function:

```
import { getProviderData, createFlagsDiscoveryEndpoint } from 'flags/next';
import * as flags from '../../../../flags';

// This function handles the authorization check for you
export const GET = createFlagsDiscoveryEndpoint(async (request) => {
  // your previous logic in here to gather your feature flags
  const apiData = await getProviderData(flags);

  // return the ApiData directly, without a NextResponse.json object.
  return apiData;
});
```

This helper function automatically handles the authorization check and adds the `x-flags-sdk-version` header where required.

▶ You can alternatively check the authorization and add the `x-flags-sdk-version` response header manually.

Next.js Pages Router

If you are using Next.js Pages Router, forward the version manually:

```
import { verifyAccess, version } from 'flags';
import { getProviderData } from 'flags/next';
import * as flags from '../../../flags';

export default async function handler(request, response) {
  const access = await verifyAccess(request.headers['authorization']);
  if (!access) return response.status(401).json(null);

  const apiData = getProviderData(flags);

  // set the required response header here, so it is
  // available on successful responses but not on the 401 above
  response.setHeader('x-flags-sdk-version', version);

  return response.json(apiData);
}
```

SvelteKit

If you are using `createHandle` in SvelteKit then the upgrade will be handled automatically for you.

If you rely on a custom `src/routes/[x+2e]well-known/vercel/flags/server.ts` file you use the new `createFlagsDiscoveryEndpoint` helper function to create your flags discovery endpoint:

```
import { createFlagsDiscoveryEndpoint, getProviderData } from 'flags/sveltekit';
import { FLAGS_SECRET } from '$env/static/private';
import * as flags from '$lib/flags';

export const GET = createFlagsDiscoveryEndpoint(
  async () => {
    // your previoius logic in here
    return getProviderData(flags);
  },
  { secret: FLAGS_SECRET },
);
```

This helper function automatically handles the authorization check and adds the `x-flags-sdk-version` header for successful responses.

Replace crypto functions

We replaced the `encrypt` and `decrypt` functions exported by the Flags SDK with dedicated functions depending on the use case.

If your app previously imported `encrypt` or `decrypt`, replace them with one of these dedicated functions depending on the type of information you are encrypting or decrypting.

- `encryptOverrides`
- `decryptOverrides`
- `encryptFlagValues`
- `decryptFlagValues`
- `encryptFlagDefinitions`
- `decryptFlagDefinitions`
- `createAccessProof`
- `verifyAccessProof`

A common example is that you would previously call `encrypt` to encrypt flag values, but you will now call the dedicated `encryptFlagValues` functions.

```
- import { encrypt, type FlagValuesType } from 'flags';
+ import { encryptFlagValues, type FlagValuesType } from 'flags';
import { FlagValues } from 'flags/react';

async function ConfidentialFlagValues({ values }: { values: FlagValuesType }) {
-  const encryptedFlagValues = await encrypt(values);
```

```
+  const encryptedFlagValues = await encryptFlagValues(values);
  return <FlagValues values={encryptedFlagValues} />;
}

export function Page() {
  const values = { exampleFlag: true };
  return (
    <div>
      {/* Some other content */}
      <Suspense fallback={null}>
        <ConfidentialFlagValues values={values} />
      </Suspense>
    </div>
  );
}
```