What does Web3Auth do?

1) Mainstream Social Account Logins, & Passwordless flows(Google, Twitter, GitHub, and any other OAuth providers)

2) Holistic Web3 Wallet/Key Management Support

: Empower users to use their wallet or key management of choice.

Using the Web3Auth suite, users can use their existing wallet or key management of choice and hook it up with your application directly.

Features

1) Non-custodial Public Key Infrastructure

2) Web and native mobile: Web3Auth works both on web as well as mobile out of the box, with native and hybrid applications supported.

Our SDKs

1) Web3Auth JavaScript SDKs for Web

- Web3Auth Plug and Play Modal SDK @web3auth/modal

- Web3Auth Plug and Play Core SDK @web3auth/core

2) Web3Auth Native SDK for Android

- web3auth-android-sdk

3) Web3Auth Native SDK for iOS

- web3auth-swift-sdk

4) Web3Auth SDK for React Native

- web3auth-react-native-sdk

5) Web3Auth SDK for Flutter

- web3auth-flutter-sdk

6) Web3Auth SDK for Unity

-web3auth-unity-sdk

--------------------------------------------------------------------------------------------------------------

Replacing Seed Phrases with Multi Factor Keys

: With Web3Auth, users handle keys similar to a multi-factor account, where they use their OAuth login

High Level Architecture

1) user triggers login

2) ID token submited to web3auth SDK (from Oauth to Web3auth SDK)

3) ID token passed for validation (from Web3auth SDK to Web3auth network Server)

4) Private key share(1/2) relayed (from Web3auth network Server to Web3auth SDK)

https://web3auth.io/docs/overview/key-management/technical-architecture/

--------------------------------------------------------------------------------------------------------------

Web3Auth for Wallets

: One of the most important use cases of Web3Auth is to integrate it within wallets.

This is because Web3Auth itself is not a wallet, it is wallet infrastructure that can be plugged into your own wallet.

Using Web3Auth to build your Wallet

1) The wallet owns the whole experience, with minimal external branding. You can even surpass that (have zero Web3Auth Branding) if you use Web3Auth Self Host.

2) You're even able to use your own OAuth 2.0 / Single Sign-On or any other flow for validating your user. You just need to provide Web3Auth with a JWT id\_token and it will handle the key generation for you.

3) You can create your wallet for any chain, depending on your needs. Web3Auth generates the private key which can be used for any chain.

4) You can even use Web3Auth to create an HDKey and generate multiple sub accounts for your users. This is useful for wallets that support multiple accounts.

5) You can choose a platform of your choice. Web3Auth SDKs are available for most mobile, web and gaming platforms.

--------------------------------------------------------------------------------------------------------------

Improvements to key recovery and redundancy

: In the event of a lost device/share, there is redundancy built into the share threshold

such that a user can still recover their key. It is also possible to refresh shares such that lost shares are revoked.

This is an improvement over writing down a seed phrase on a piece of paper,

since losing the seed phrase gives complete access to the private key. Losing a share,

however, is acceptable as long as the user does not lose more than one share without refreshing his existing shares.

Powered by the Auth Network

In no particular order, current node operators are:

1) Binance

2) Ethereum Name Service

3) Etherscan

4) Polygon (MATIC)

5) Zilliqa

6) Tendermint

7) Ontology

8) SKALE

9) Torus

--------------------------------------------------------------------------------------------------------------

Key Assignments

- The keys are assigned to a combination of verifier (e.g., Google, Reddit, Discord) and verifier\_id (e.g., email, username),

which is a unique identifier respective to and provided by the verifier.‌ This assignment can be triggered by any node

and is decided through the nodes consensus layer.

Verifiers and Key Retrieval

1. Your application gets the user to sign-in via their preferred method (OAuth / email password / passwordless / verification code).

2. After the user gives consent/verifies his/her email, Torus SDK will receive an ID token and assign a key to the user depending on User Verifier ID from ID Token.

3. The key is retrieved from the Torus network and exposed to Web3 provider (DApp) to complete user sign-in request.

4. Torus uses this ID Token to check if the user’s profile information exists in the DApp.

- If it does, the user will be signed in to the DApp with their preferred login.

- If it doesn’t, the user can create a new account on the DApp with their preferred login.

--------------------------------------------------------------------------------------------------------------

[evm] Web3Auth x React Demo Application

https://github.com/Web3Auth/examples/tree/main/web-modal-sdk/evm/react-evm-web3auth-example

BFT network

--------------------------------------------------------------------------------------------------------------

[Developer Dashboard]

Enable Interoperability

<https://web3auth.io/docs/developer-dashboard/enable-interoperability>

<https://web3auth.io/docs/interoperability>

--------------------------------------------------------------------------------------------------------------

Web3Auth Self Host SDK

For Self Hosting, we give you the option to directly try out our open source, UI-less SDK, tKey.

The tKey SDK is the underlying SDK used to implement Web3Auth Plug n Play.

tKey stands for Threshold Key, which refers to the management keys and shares generated using threshold cryptography.

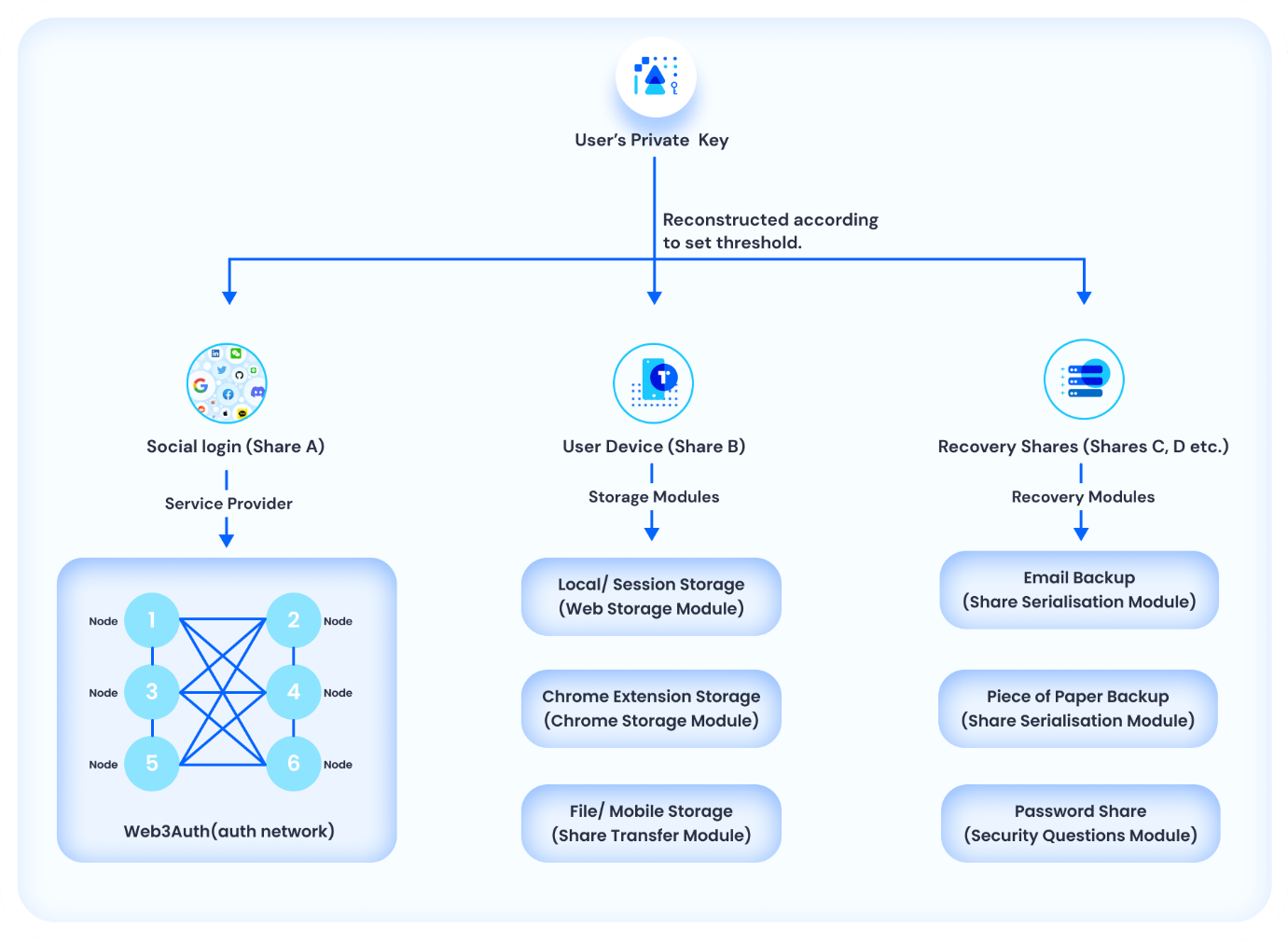
The tKey SDK manages private keys by generating shares using Shamir Secret Sharing.

For a 2 out of 3 (2/3) setup, we give the user three shares: ShareA, ShareB, and ShareC.

- ShareA is managed and split across Web3Auth's Auth Network, accessed by an OAuth login provider that a user owns. For example, a user could use their Google account to access their share.

- ShareB is stored on the user’s device: The implementation is device and system specific. For example, on mobile devices, this share could be stored in device storage secured via biometrics.

- ShareC is a recovery share: An additional share to be kept by the user, possibly kept on a separate device, downloaded, or based on user input with enough entropy (e.g., password, security questions, hardware device, etc.).



https://web3auth.io/docs/self-hosting

--------------------------------------------------------------------------------------------------------------

Connect your Blockchain

https://web3auth.io/docs/connect-blockchain/

Web3Auth is the frontend authentication system for your dApp.

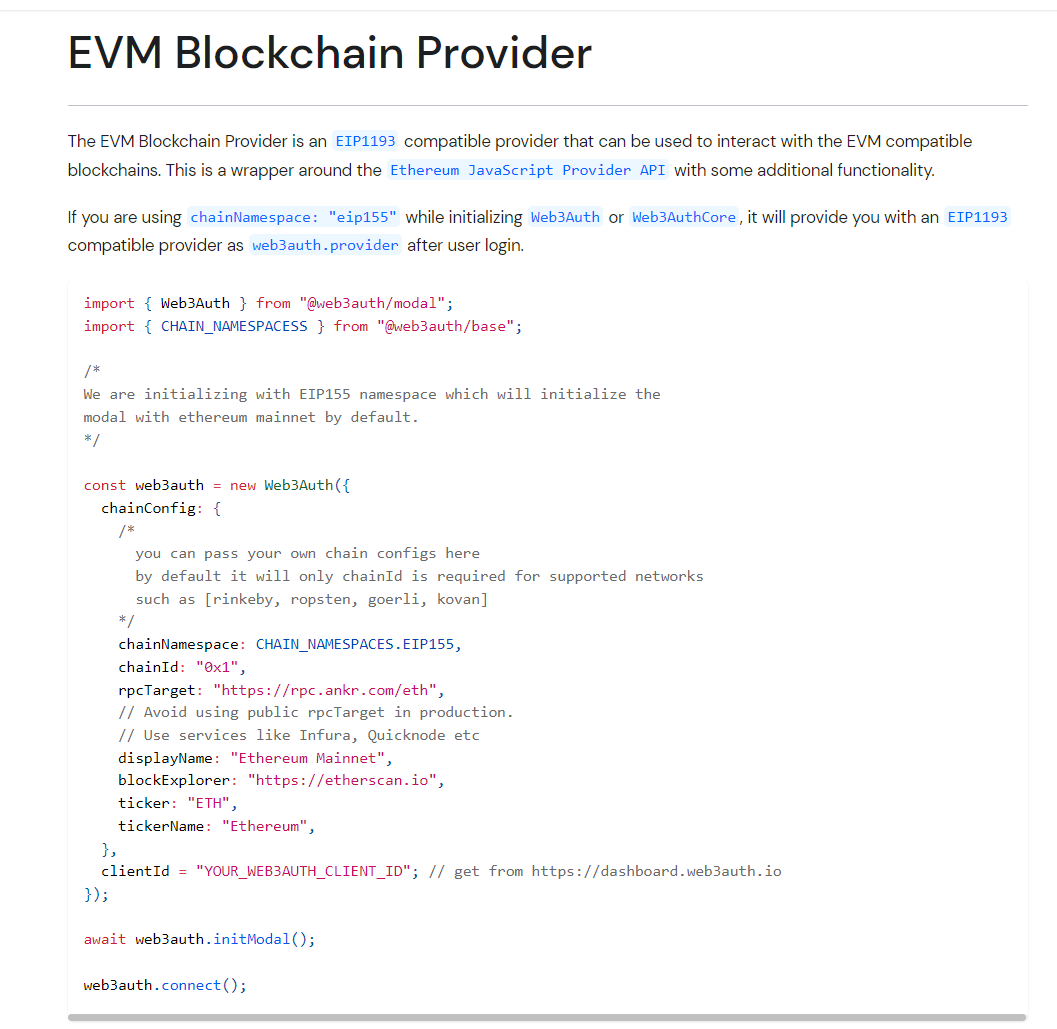
Once the user is authenticated, the Web3Auth SDK returns a "provider".

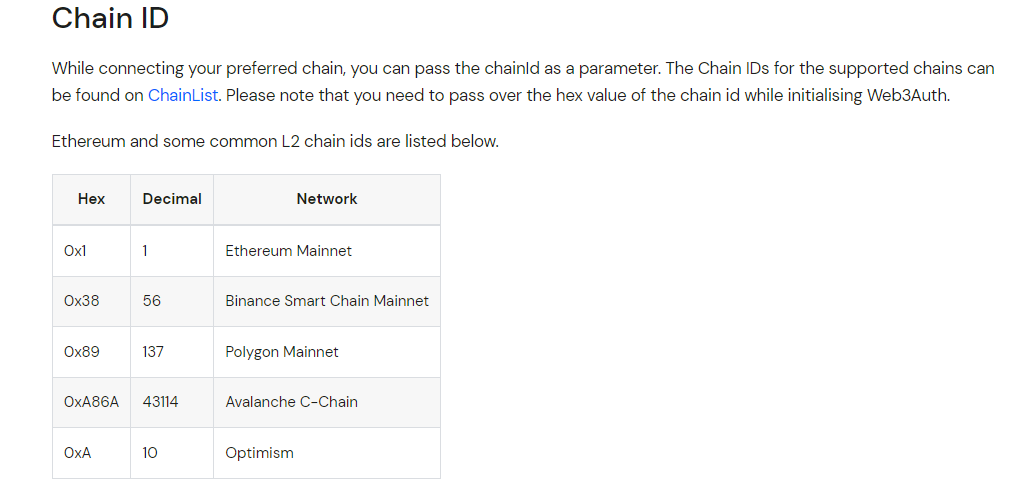
Using this provider we can sign transactions and make RPC(remote procedure call) calls to any blockchain.

Using Web3Auth Providers

A provider is how libraries like web3.js & ethers.js talk to the blockchain. Providers take JSON-RPC requests and return the response. This is normally done by submitting the request to an HTTP or IPC socket-based server. Each adapter in Web3Auth exposes a Provider on successful user authentication. This provider can be used to interact with wallet or connected chain using RPC calls. Currently, Web3Auth supports providers for both **EVM** and **Solana** chains, which means that any EVM compatible chain & Solana can directly use our native providers to interact with the dApp. You can follow our API Reference to know more about how to use these providers with their compatible adapters:

**EVM Blockchain Provider**





<https://web3auth.io/docs/sdk/web/providers/evm>

**Get User account and Balance**[**​**](https://web3auth.io/docs/sdk/web/providers/evm#get-user-account-and-balance)

**web3.eth.getAccounts()**[**​**](https://web3auth.io/docs/sdk/web/providers/evm#web3ethgetaccounts)

This method is used to fetch the address of the connected account.

import Web3 from "web3";  
  
// refer to `Getting a provider with web3auth login.` section above  
// to know how `web3auth.provider` is initialized after user's login  
const web3 = new Web3(web3auth.provider);  
  
// Get user's Ethereum public address  
const address = (await web3.eth.getAccounts())[0];  
  
// Get user's balance in ether  
const balance = web3.utils.fromWei(  
 await web3.eth.getBalance(address) // Balance is in wei  
);

**Send Transaction**[**​**](https://web3auth.io/docs/sdk/web/providers/evm#send-transaction)

#### web3.eth.sendTransaction(object)[​](https://web3auth.io/docs/sdk/web/providers/evm#web3ethsendtransactionobject)

This function is used to broadcast a transaction on chain.

import Web3 from "web3";  
// refer to `Getting a provider with web3auth login.` section above  
// to know how `web3auth.provider` is initialized after user's login  
const web3 = new Web3(web3auth.provider);  
  
// Get user's Ethereum public address  
const fromAddress = (await web3.eth.getAccounts())[0];  
  
const destination = "0xE0cef4417a772512E6C95cEf366403839b0D6D6D";  
const amount = web3.utils.toWei(1); // Convert 1 ether to wei  
  
// Submit transaction to the blockchain and wait for it to be mined  
const receipt = await web3.eth.sendTransaction({  
 from: fromAddress,  
 to: destination,  
 value: amount,  
});

**Sign a message**[**​**](https://web3auth.io/docs/sdk/web/providers/evm#sign-a-message)

The following example shows how to sign different types of messages with the connected user's private key.

#### Personal Sign[​](https://web3auth.io/docs/sdk/web/providers/evm#personal-sign)

#### web3.eth.personal.sign(originalMessage, fromAddress)[​](https://web3auth.io/docs/sdk/web/providers/evm#web3ethpersonalsignoriginalmessage-fromaddress)

import Web3 from "web3";  
  
// refer to `Getting a provider with web3auth login.` section above  
// to know how `web3auth.provider` is initialized after user's login  
const web3 = new Web3(web3auth.provider);  
  
// Get user's Ethereum public address  
const fromAddress = (await web3.eth.getAccounts())[0];  
  
const originalMessage = "YOUR\_MESSAGE";  
  
const signedMessage = await web3.eth.personal.sign(originalMessage, fromAddress);

**Sign Typed Data V1**[**​**](https://web3auth.io/docs/sdk/web/providers/evm#sign-typed-data-v1)

#### eth\_signTypedData[​](https://web3auth.io/docs/sdk/web/providers/evm" \l "eth_signtypeddata" \o "Direct link to eth_signtypeddata)

import Web3 from "web3";  
  
// refer to `Getting a provider with web3auth login.` section above  
// to know how `web3auth.provider` is initialized after user's login  
const web3 = new Web3(web3auth.provider);  
  
// Get user's Ethereum public address  
const fromAddress = (await web3.eth.getAccounts())[0];  
  
const originalMessage = [  
 {  
 type: "string",  
 name: "fullName",  
 value: "John Doe",  
 },  
 {  
 type: "uint32",  
 name: "userId",  
 value: "1234",  
 },  
];  
const params = [originalMessage, fromAddress];  
const method = "eth\_signTypedData";  
  
const signedMessage = await web3.currentProvider.sendAsync({  
 id: 1,  
 method,  
 params,  
 fromAddress,  
});

**Fetch User's Private Key**[**​**](https://web3auth.io/docs/sdk/web/providers/evm#fetch-users-private-key)

#### eth\_private\_key[​](https://web3auth.io/docs/sdk/web/providers/evm" \l "eth_private_key" \o "Direct link to eth_private_key)

This method is used to fetch the private key of logged in user. It is only available for in-app adapters like openlogin.

//Assuming user is already logged in.  
async getPrivateKey() {  
 const privateKey = await web3auth.provider.request({  
 method: "eth\_private\_key"  
 });  
 //Do something with privateKey  
}

**Additional Reading**[**​**](https://web3auth.io/docs/sdk/web/providers/evm#additional-reading)

### **Getting a provider from any secp256k1 private key**[**​**](https://web3auth.io/docs/sdk/web/providers/evm#getting-a-provider-from-any-secp256k1-private-key)

@web3auth/ethereum-provider package can be used to get an [EIP1193](https://eips.ethereum.org/EIPS/eip-1193) compatible provider from any secp256k1 private key. You can use this package with our SDKs, to get an [EIP1193](https://eips.ethereum.org/EIPS/eip-1193) provider from the logged-in user's private key.

#### Example[​](https://web3auth.io/docs/sdk/web/providers/evm#example)

import { EthereumPrivateKeyProvider } from "@web3auth/ethereum-provider";  
  
/\*  
privKey: any secp256k1 private key.  
\*/  
  
async setProvider(privKey: string) {  
 const ethereumPrivateKeyProvider = new EthereumPrivateKeyProvider({  
 config: {  
 /\*  
 pass the chain config that you want to connect with  
 all chainConfig fields are required.  
 \*/  
 chainConfig: {  
 chainId: "0x1",  
 rpcTarget: `https://rpc.ankr.com/eth`,  
 displayName: "ropsten",  
 blockExplorer: "https://ropsten.etherscan.io/",  
 ticker: "ETH",  
 tickerName: "Ethereum",  
 },  
 },  
 });  
 /\*  
 pass user's private key here.  
 after calling setupProvider, we can use  
 \*/  
 await ethereumPrivateKeyProvider.setupProvider(privKey);  
 console.log(ethereumPrivateKeyProvider.provider);  
}

# **Torus Wallet Connector Plugin**

## Adding JSON RPC APIs[​](https://web3auth.io/docs/connect-blockchain/#adding-json-rpc-apis)

## Web3Auth providers give you a standard way of interacting with the blockchain. However, alongside that, it is recommended to use JSON RPC APIs, which help you connect to the blockchain without the need to run your own instance/ nodes. There are many services which offer a web API for accessing different blockchains:

#### Infura[​](https://web3auth.io/docs/connect-blockchain/" \l "infura" \o "Direct link to Infura)

[Infura](https://infura.io/) is the leading platform for Ethereum infrastructure. It provides a gateway to the Ethereum network, allowing developers to build and scale decentralized applications without having to run their own infrastructure. Most of the Web3Auth backend infrastructure runs on Infura APIs.

#### Quicknode[​](https://web3auth.io/docs/connect-blockchain/" \l "quicknode" \o "Direct link to Quicknode)

[Quicknode](https://www.quicknode.com/) is a managed blockchain node service that provides high-performance access to 15+ blockchains, including Ethereum, Gnosis (xDAI), Polygon, Binance Smart Chain, Avalanche, Fantom, Solana, Optimism, Arbitrum (+Nova), Algorand, Harmony, Celo, Terra and Bitcoin networks. They're the biggest node providers for the Solana Ecosystem, and our major partners for Solana infrastructure.

#### Alchemy[​](https://web3auth.io/docs/connect-blockchain/#alchemy)

As a developer platform, [Alchemy](https://www.alchemy.com/) provides a suite of developer toolings and abstractions including JSON RPC APIs across all major chains, an ethers.js SDK, and a library of [enhanced APIs](https://docs.alchemy.com/reference/enhanced-apis-overview) like their NFT APIs.

#### Ankr[​](https://web3auth.io/docs/connect-blockchain/" \l "ankr" \o "Direct link to Ankr)

[Ankr](https://www.ankr.com/) is a decentralized cloud computing platform that provides a full suite of infrastructure services for blockchain developers. Most of the examples in our documentation use Ankr's Public JSON RPC APIs, to help you kickstart.

--------------------------------------------------------------------------------------------------------------

# **Integrate Web3Auth with Multiple Chains**

The Web3Auth Web SDK supports all blockchains following the secp256k1 & ed25519 curves. This is because Web3Auth focuses of generating keys non custodially, and using the finally generated key one can use the corresponding libraries of the blockchain to make calls like getting user's account, fetch balance, sign transaction, send transaction, read from and write to the smart contract, etc.

Not just that, you can connect multiple chains at once and use them simultaneously. We have highlighted a few methods for a few chains here for getting you started quickly on that.

NOTE

This reference is for the Web, however, you can use all the blockchains on Android, iOS, React Native, Flutter, & Unity as well. Please follow our reference for Ethereum, and similarly use corresponding blockchain libraries that support the platforms to use the private key and make blockchain calls accordingly.

--------------------------------------------------------------------------------------------------------------

# **Using Custom Authentication**

Custom Authentication refers to the capability of hooking up your own custom authentication flow with Web3Auth. We recommend you to use Custom Authentication rather than relying on the default authentication Modal provided by Web3Auth.

Upsides to using it are:

- Provides holistic user experience from start to finish.

- Customise the data needed from the authentication service and onboard users according to your requirements.

- Migrate your existing users into the new auth flow with Web3Auth, without them needing to create new accounts.

# **Creating Verifiers on the Web3Auth Dashboard**

#### What is a Verifier?[​](https://web3auth.io/docs/custom-authentication/verifiers#what-is-a-verifier)

A verifier is a piece of information about the OAuth provider being used by the application, which is queried by web3auth auth network nodes from a smart contract deployed on the Ethereum blockchain to verify the JWT token.

There are two types of Verifiers at Web3Auth:

1. Default Verifiers: These are the verifiers associated with OAuth providers(Google, Facebook, Apple etc) which are owned and managed by Web3Auth's account.
2. Custom Verifiers: If you want to manage your OAuth providers(Google, Facebook, Apple etc) yourself or to use some Federated Identity Providers like Auth0, Firebase, AWS Cognito, Okta etc, or even your Custom JWT Authentication, you'll need to create a verifier on the Web3Auth's dashboard.

Custom Verifier is needed to let the user login with your custom authentication flow. The Public/Private keys generated for a user are scoped to a verifier. Verifier scripts determine the access structure to your user's keys, and the set Verifier scripts are chosen to determine which logins your application will ultimately use.

You can read more about verifiers [in this blog post](https://medium.com/toruslabs/key-assignments-resolution-and-retrieval-afb984500612) or [in this GitHub discussion](https://github.com/orgs/Web3Auth/discussions/427#discussioncomment-3442052).

The Custom Auth Dashboard looks something like this:

### **Select Environment**[**​**](https://web3auth.io/docs/custom-authentication/verifiers#select-environment)

Select between testnet, mainnet, aqua, celeste and cyan.

* testnet is a sandbox environment for developers to experiment. People usually test and finish their integration here.
* mainnet, aqua, celeste & cyan are the production environments for scalable applications.

### **Login Provider**[**​**](https://web3auth.io/docs/custom-authentication/verifiers#login-provider)

Select the Login Provider from the list to create a verifier of that type.

The following are the list we have on our Dashboard, if something you're loooking is not available in the list, choose Custom and provide the JWT details.

--------------------------------------------------------------------------------------------------------------

# **Whitelabeling your Web3Auth Integration**

<https://web3auth.io/docs/whitelabel/login-modal>

<https://web3auth.io/docs/whitelabel/user-flow-screens>

<https://web3auth.io/docs/whitelabel/torus-wallet-plugin>

const login = async () => {

    if (!web3auth) {

      uiConsole("web3auth not initialized yet");

      return;

    }

    const web3authProvider = await web3auth.connectTo(

      WALLET\_ADAPTERS.OPENLOGIN,

      {

        loginProvider: "google",

      }

    );

    setProvider(web3authProvider);

  };