Onboard Users to Developer-Controlled Wallets

Streamline user onboarding for developer-controlled wallets Suggest Edits

Developer-controlled Programmable Wallets simplify user onboarding to your Web3 app by performing blockchain interactions, such as asset transfers and smart contract executions, on behalf of your users. You maintain control of the wallet while removing all of the complexity for your users.

This guide describes the best practices for creating and managing developer-controlled wallets to integrate into your app, including how to create wallets for existing users already using your app and new users who sign up later.

Scenarios:

1. Create wallets for existing users:
2. For onboarding your existing users to developer-controlled wallets, ensuring a smooth transition to the Web3 ecosystem.
3. Create wallets for new users in advance:
4. For the routine onboarding of new users to your application. You can create wallets in advance and assign them to users when they sign up.

Cost Consideration for Externally Owned Account (EOA) Wallets

Creating EOA wallets on mainnet incurs a minor expense per wallet. To optimize costs, especially when planning to create a large number of wallets, it's prudent to carefully predict user demand and batch-create wallets accordingly. For instance, forecast and prepare wallets to meet the expected demand in the next 30 days to avoid unnecessary expenses and ensure optimal resource utilization.

Prerequisites

1. Ensure you complete theCreate Your First Developer-Controlled Wallet
2. guide. It provides a comprehensive walkthrough for setting up your first entity secret, wallet set, and wallet, making it an essential foundation for understanding developer-controlled wallets.

Create wallets for existing users

1. Batch create and link wallets to existing users:
2. Use the
3. count
4. parameter with thePOST /developer/wallets
5. API endpoint to specify the number of wallets needed per blockchain. Simultaneously, link these wallets to individual users by utilizing the
6. metadata
7. attribute to designate wallet names and associate them with user IDs (e.g., UUID) using the
8. refId
9. field for each wallet.

You can create up to 20 wallets per request. If you need more, make additional requests until the required total is reached. Node.js cURL // Import and configure the developer-controlled wallet SDK const { initiateDeveloperControlledWalletsClient } = require('@circle-fin/developer-controlled-wallets'); const circleDeveloperSdk = initiateDeveloperControlledWalletsClient({ apiKey: '', entitySecret: '' });

const response = await circleDeveloperSdk.createWallets({ walletSetId: '71f2a6b4-ffa7-417a-ad5b-fb928753edc8', accountType: 'SCA', blockchains: ['MATIC-MUMBAI'], count: 2, metadata: [{ name: 'User Wallet1', refId: 'UUID1', }, { name: 'User Wallet2', refId: 'UUID2', },], }); curl --request POST \ --url https://api.circle.com/v1/w3s/developer/wallets \ --header 'accept: application/json' \ --header 'content-type: application/json' \ --header 'authorization: Bearer ' \ --data ' { "idempotencyKey": "a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11", "walletSetId": "71f2a6b4-ffa7-417a-ad5b-fb928753edc8" "accountType": "SCA", "blockchains": ["MATIC-MUMBAI"], "entitySecretCiphertext": "0NtD3d3+nmgb4GqYQXzAjKF8h5Zq6sHM2k/7UQz5OIXWnU7H8O9XvVYXqDnw1wEmY...", "count": 2, "metadata": [{ "name": "User Wallet1", "refId": "UUID1" }, { "name": "User Wallet2", "refId": "UUID2" }] } ' 1. Inspect the response payload: 2. After receiving a successful response, inspect the response body and map the wallet IDs 3. data.wallets.id 4. with the respective users on your systems. The response body includes information such as the wallet IDs, custody types, names, and reference IDs.

Response Body { "data": { "wallets": [{ "id": "66b67097-307e-5b46-a1d5-0b1577d67fd4", "state": "LIVE", "walletSetId": "018b42d2-cc38-7a1e-a47a-5927d2c97f16", "custodyType": "DEVELOPER", "refId": "UUID1", "name": "User Wallet1", "address": "0xe55628c98f5d81daaa79b72899b38a3535d10990", "blockchain": "MATIC-MUMBAI", "accountType": "SCA", "updateDate": "2024-01-22T22:57:20Z", "createDate": "2024-01-22T22:57:20Z" }, { "id": "cda61d8d-7d46-5a39-a8a6-6b4a3d6fdac3", "state": "LIVE", "walletSetId": "018b42d2-cc38-7a1e-a47a-5927d2c97f16", "custodyType": "DEVELOPER", "refId": "UUID2", "name": "User Wallet1", "address": "0x29b27e792b8b854e48e85ab4f456cf5a9f1579fb", "blockchain": "MATIC-MUMBAI", "accountType": "SCA", "updateDate": "2024-01-22T22:57:20Z", "createDate": "2024-01-22T22:57:20Z" }] } } By successfully following these steps, you can create wallets for existing users and link them accordingly.

Create wallets for users in advance

Creating wallets in advance streamlines the real-time onboarding process and prevents delays when users require wallets. This approach optimizes the workflow and ensures that wallets are readily available whenever users need them.

The following demonstrates best practices:

1. Invoke the developer-controlled wallets creation API:
2. Utilize thePOST: /developer/wallets
3. endpoint for wallet creation. Specify the number of wallets needed by adjusting the
4. count
5. parameter.

You can create up to 20 wallets per request. If you need more, make additional requests until the required total is reached. Node.js cURL // Import and configure the developer-controlled wallet SDK const { initiateDeveloperControlledWalletsClient } = require('@circle-fin/developer-controlled-wallets'); const circleDeveloperSdk = initiateDeveloperControlledWalletsClient({ apiKey: '', entitySecret: '' });

const response = await circleDeveloperSdk.createWallets({ walletSetId: '71f2a6b4-ffa7-417a-ad5b-fb928753edc8', accountType: 'SCA', blockchains: ['MATIC-MUMBAI'], count: 2 }); curl --request POST \ --url https://api.circle.com/v1/w3s/developer/wallets \ --header 'accept: application/json' \ --header 'content-type: application/json' \ --header 'authorization: Bearer ' \ --data ' { "idempotencyKey": "a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11", "walletSetId": "71f2a6b4-ffa7-417a-ad5b-fb928753edc8" "accountType": "SCA", "blockchains": ["MATIC-MUMBAI"], "entitySecretCiphertext": "0NtD3d3+nmgb4GqYQXzAjKF8h5Zq6sHM2k/7UQz5OIXWnU7H8O9XvVYXqDnw1wEmY...", "count": 2 } ' Response Body { "data": { "wallets": [{ "id": "66b67097-307e-5b46-a1d5-0b1577d67fd4", "state": "LIVE", "walletSetId": "018b42d2-cc38-7a1e-a47a-5927d2c97f16", "custodyType": "DEVELOPER", "address": "0xe55628c98f5d81daaa79b72899b38a3535d10990", "blockchain": "MATIC-MUMBAI", "accountType": "SCA", "updateDate": "2024-01-22T22:57:20Z", "createDate": "2024-01-22T22:57:20Z" }, { "id": "cda61d8d-7d46-5a39-a8a6-6b4a3d6fdac3", "state": "LIVE", "walletSetId": "018b42d2-cc38-7a1e-a47a-5927d2c97f16", "custodyType": "DEVELOPER", "address": "0x29b27e792b8b854e48e85ab4f456cf5a9f1579fb", "blockchain": "MATIC-MUMBAI", "accountType": "SCA", "updateDate": "2024-01-22T22:57:20Z", "createDate": "2024-01-22T22:57:20Z" }] } } 1. Store wallet information: 2. Post-creation, ensure that all relevant wallet information, including wallet IDs and associated attributes (e.g., blockchains, custody types), is securely stored in your system for future reference. 3. Assign the wallet to the user: 4. Once a user requests a wallet, assign one of the wallets created in the previous step to them. This assignment process should occur on your system and within Circle's infrastructure. 5. 1. Identify an available pre-created wallet: 6. 2. Select a wallet from your pool of created wallets that hasn't been assigned yet. 7. 3. Associate the selected wallet with the user: 8. 4. Update your database to link the user's account with the selected wallet. This link should include the wallet's ID, blockchain information, and any other relevant details to ensure that transactions can be accurately processed and attributed to the correct user. 9. 5. Mark the wallet as assigned: 10. 6. To prevent the same wallet from being assigned to multiple users, update its status in your database to indicate that it has now been assigned. 11. 7. Update Circle’s records associating the user and the assigned wallet: 12. 8. Use thePUT: /developer/wallets/{id} 13. 9. API to update Circle’s records.

Node.js cURL const response = await circleDeveloperSdk.updateWallet({ id: '66b67097-307e-5b46-a1d5-0b1577d67fd4', name: 'User Wallet1', refId: 'UUID1' }); curl --request PUT \ --url https://api.circle.com/v1/w3s/developer/wallets/66b67097-307e-5b46-a1d5-0b1577d67fd4 \ --header 'accept: application/json' \ --header 'content-type: application/json' \ --header 'authorization: Bearer ' \ --data ' { "id": "66b67097-307e-5b46-a1d5-0b1577d67fd4", "name": "User Wallet1", "refId": "UUID1" } ' Response Body { "data": { "wallet": { "id": "66b67097-307e-5b46-a1d5-0b1577d67fd4", "state": "LIVE", "walletSetId": "018b42d2-cc38-7a1e-a47a-5927d2c97f16", "custodyType": "DEVELOPER", "refId": "UUID1", "name": "User Wallet1", "address": "0xe55628c98f5d81daaa79b72899b38a3535d10990", "blockchain": "MATIC-MUMBAI", "accountType": "SCA", "updateDate": "2024-01-22T22:57:20Z", "createDate": "2024-01-22T22:57:20Z" } } } By adhering to these guidelines, you can streamline the wallet management process, ensuring that wallets are efficiently assigned to users on an as-needed basis. Updatedabout 2 months ago * Table of Contents * * Prerequisites * * Create wallets for existing users * * Create wallets for users in advance