CIRCLE

Implementation Guide

Circle Modular Smart Contract Accounts

Version 1.7

Table of Contents

Version History

#	Date	Change	Author
1.0	Aug 19, 2024	Initial draft completed	Allison Kaufman
1.1	Sep 5, 2024	Updated contract address	Patrick Phua
1.2	Oct 23, 2024	Updated contract address for v2	Patrick Phua
1.3	Nov 20, 2024	Moving modules to external docs	Patrick Phua
1.4	Feb 11, 2025	Updated contract address for v3	Patrick Phua
1.5	Feb 12, 2025	Added link to latest Security Audit	Ritesh Patel
1.6	May 20, 2025	Updated contract address for ColdStorageAddressBookPlugin	Patrick Phua
1.7	Dec 1, 2025	Updated Circle documentation links	Alex Lewis

Introduction

This document outlines the steps for

1. setting up a Circle Developer account,
2. creating a Programmable Wallets Modular Smart Contract Account (MSCA), and,
3. transferring control over the created wallet to an externally-managed key

After completing the steps, you will have an MSCA wallet on Ethereum's Sepolia Testnet with ownership transferred to your externally-managed key. We have also included some test transfer steps to validate that the ownership transfer was successful.

At the bottom of the document, we have included a number of additional resources, including our API reference and public documentation, Postman collections, and interactive quick start guides, to assist with the setup process and execution of requests. You can also find the contract address for the MSCA factory contract.

Programmable Wallets MSCA Creation

Step 1: Set up a Web3 Console Account

1. Sign up for account
2. Create API key
3. Generate entity secret
    a. Generate a 32-byte string value

None

```
openssl rand -hex 32
```

    b. Fetch your entity's public key

None

```
curl --request GET \
    --url 'https://api.circle.com/v1/w3s/config/entity/publicKey' \
    --header 'accept: application/json' \
    --header 'authorization: Bearer <API_KEY>'
```

    c. Generate the Entity Secret Ciphertext

Python

```
import base64
import codecs
# Installed by `pip install pycryptodome`
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Hash import SHA256

# Paste your entity public key here.
public_key_string = 'PASTE_YOUR_PUBLIC_KEY_HERE'

# If you already have a hex encoded entity secret, you can paste it here. the length of the hex string should be 64.
```

```
hex_encoded_entity_secret = 'PASTE_YOUR_HEX_ENCODED_ENTITY_SECRET_KEY_HERE'

# The following sample codes generate a distinct entity secret ciphertext with each execution.
if __name__ == '__main__':
    entity_secret = bytes.fromhex(hex_encoded_entity_secret)

    if len(entity_secret) != 32:
        print("invalid entity secret")
        exit(1)

    public_key = RSA.importKey(public_key_string)

    # encrypt data by the public key
    cipher_rsa = PKCS1_OAEP.new(key=public_key, hashAlgo=SHA256)
    encrypted_data = cipher_rsa.encrypt(entity_secret)

    # encode to base64
    encrypted_data_base64 = base64.b64encode(encrypted_data)

    print("Hex encoded entity secret:", codecs.encode(entity_secret, 'hex').decode())
    print("Entity secret ciphertext:", encrypted_data_base64.decode())
```

d. Register the Entity Secret Ciphertext
   i. Access the Configurator Page in the developer console
   ii. Enter the Entity Secret Ciphertext generated in the previous step
   iii. Select "Register" to complete the Entity Secret Ciphertext registration

Step 2: Create Programmable Wallets MSCA and Test

1. Create a wallet set
    a. Note that you will need to re-run the code from **Step 1 - 3c** above to create a new Entity Secret Ciphertext for each API request that requires this parameter. More details can be found on this page.

None

```
curl --request POST \
    --url 'https://api.circle.com/v1/w3s/developer/walletSets' \
    --header 'accept: application/json' \
    --header 'content-type: application/json' \
    --header 'authorization: Bearer <API_KEY>' \
    --data '
{
  "idempotencyKey": "<UUID_V4>",
  "name": "Entity WalletSet A",
  "entitySecretCiphertext": "<ENTITY_SECRET_CIPHERTEXT>"
}
'
```

2. Create a wallet
    a. Ensure that the accountType is set to “SCA”

None

```
curl --request POST \
    --url 'https://api.circle.com/v1/w3s/developer/wallets' \
    --header 'accept: application/json' \
    --header 'content-type: application/json' \
    --header 'authorization: Bearer <API_KEY>' \
    --data '
{
  "idempotencyKey": "<UUID_V4>",
 "accountType": "SCA",
 "blockchains": [
   "ETH-SEPOLIA"
 ],
 "count": 1,
 "entitySecretCiphertext": "<ENTITY_SECRET_CIPHERTEXT>",
 "walletSetId": "<WALLET_SET_ID>"
}
'
```

3. Get testnet USDC and Sepolia ETH in your wallet from a faucet or other source
4. Test transfer of tokens (USDC or native) from wallet using PWs API.
   Note that this step is required for the MSCA to be fully deployed on-chain.
   a. Check wallet's balance

None

```
curl --request GET \
    --url 'https://api.circle.com/v1/w3s/wallets/{id}/balances' \
    --header 'accept: application/json' \
    --header 'authorization: Bearer <API_KEY>'
```

   b. Transfer tokens

None

```
curl --request POST \
    --url 'https://api.circle.com/v1/w3s/developer/transactions/transfer' \
    --header 'accept: application/json' \
    --header 'content-type: application/json' \
    --header 'authorization: Bearer <API_KEY>' \
    --data '
{
 "idempotencyKey": "<UUI_V4>",
 "walletId": "<ORIGIN_WALLET_ID>",
 "tokenId": "<TOKEN_ID>",
 "destinationAddress": "<DESTINATION_ADDRESS>",
 "amounts": [
   ".01"
 ],
 "feeLevel": "MEDIUM",
 "entitySecretCiphertext": "<ENTITY_SECRET_CIPHERTEXT>"
}
'
```

c. [Check transfer state]

```
None
curl --request GET \
     --url 'https://api.circle.com/v1/w3s/transactions/{id}' \
     --header 'accept: application/json' \
     --header 'authorization: Bearer <API_KEY>'
```

d. Verify the wallet address on chain (e.g. via Etherscan)

Step 3: Change Ownership to Externally-Managed EOA and Test

1. Change ownership to externally-managed EOA

None

```
curl --request POST \
     --url
'https://api.circle.com/v1/w3s/developer/transactions/contractExecution' \
     --header 'accept: application/json' \
     --header 'authorization: Bearer <API_KEY>' \
     --header 'content-type: application/json' \
     --data '
{
  "abiParameters": [
    "<EXTERNAL_OWNER_ADDRESS>"
  ],
  "idempotencyKey": "<UUID_V4>",
  "abiFunctionSignature": "transferNativeOwnership(address)",
  "contractAddress": "<MSCA_ADDRESS>",
      // the above <MSCA_ADDRESS> is your PW generated SCA wallet address
  "refId": "Transfer ownership to External Wallet",
  "walletId": "<WALLET_ID>",
  "entitySecretCiphertext": "<ENTITY_SECRET_CIPHERTEXT>",
  "feeLevel": "MEDIUM"
}'
```

2. Test transfer with EOA new owner
    a. Example uses cast from Foundry
    b. Ensure that you have sufficient Sepolia ETH at the controlling EOA to cover network fees

None

```
cast send --gas-limit 1000000 --private-key $OWNER_PRIVATE_KEY --rpc-url
$RPC_URL $MSCA_ADDRESS "execute(address,uint256,bytes)" $USDC_CONTRACT_ADDRESS
0 $(cast calldata "transfer(address,uint256)" $RECIPIENT_ADDRESS 1)
```

3. Confirm that the transaction completed as expected.

   a. You can check your Circle Developer account to see the outbound transaction amount or you can view your transaction directly on Etherscan
   b. Note: Gas will be paid by the controlling EOA

4. Confirm that the MSCA is no longer controlled by the Circle-managed key
   a. Initiate a transfer via the Programmable Wallets API - this transfer should **fail** if ownership has been correctly transferred

None

```
curl --request POST \
     --url 'https://api.circle.com/v1/w3s/developer/transactions/transfer'
\
     --header 'accept: application/json' \
     --header 'content-type: application/json' \
     --header 'authorization: Bearer <API_KEY>' \
     --data '
{
  "idempotencyKey": "<UUI_V4>",
  "walletId": "<ORIGIN_WALLET_ID>",
  "tokenId": "<TOKEN_ID>",
  "destinationAddress": "<DESTINATION_ADDRESS>",
  "amounts": [
    ".01"
  ],
  "feeLevel": "MEDIUM",
  "entitySecretCiphertext": "<ENTITY_SECRET_CIPHERTEXT>"
}
'
```

   b. At this point, you can transfer your USDC into your MSCA

Deploying to Mainnet

1. After completing the above steps on testnet, the next step is to upgrade from testnet to mainnet following the process outlined in this guide.
2. After upgrading to mainnet, we recommend that you configure a gas policy (required to create a MSCA).

Additional Resources

Contract Details

Name	Contract Address	Github	Audit
circle_6900_singleowner_v3	Mainnet factory address: 0xf61023061ed45fa9eAC4D2670649cE1FD37ce536 Mainnet implementation address: 0xD206aC7fEf53d83ED4563E770b28Dba90D0D9eC8 Testnet factory address: 0xf61023061ed45fa9eAC4D2670649cE1FD37ce536 Testnet implementation address: 0xD206aC7fEf53d83ED4563E770b28Dba90D0D9eC8	circlefin / buidl-wallet-contracts	Chain…

Updating MSCA

MSCA wallets will be upgradeable as Circle continues to develop additional functionality for the accounts.

Below is an example of how to perform an upgrade to the core MSCA functionality.

```
None
$CALL_DATA = cast abi-encode "SingleOwnerMSCA.initializeSingleOwnerMSCA(address)"
$OWNER_ADDRESS

cast send --private-key $OWNER_PRIVATE_KEY $MSCA_ADDRESS
"upgradeToAndCall(address,bytes)" "[$MSCA_IMPL_ADDRESS $CALL_DATA]" --rpc-url $RPC_URL
```

API Resources

- Complete API Reference
- Postman Collection for Developer Controlled Programmable Wallets
- Quick start guide to creating a Developer-Controlled Wallet
- Interactive guide with code samples for a more illustrative walkthrough. You can follow Steps 1-2 in this guide to create your Wallet Set. When you get to Step 2.2, you will need to execute the API call separately from the guide to create wallets on Ethereum Sepolia testnet
- More information about Account Types