Solana Programs

CCTP Programs on the Solana blockchain Suggest Edits

Overview

Solana CCTP programs are written in Rust and leverage the Anchor framework. The Solana CCTP protocol implementation is split into two programs: MessageTransmitter and TokenMessengerMinter . TokenMessengerMinter encapsulates the functionality of both TokenMessenger and TokenMinter contracts on EVM chains. To ensure alignment with EVM contracts' logic and state, and to facilitate future upgrades and maintenance, the code and state of Solana programs reflect the EVM counterparts as closely as possible.

Devnet Program Addresses

Program Domain Address MessageTransmitter 5 CCTPmbSD7gX1bxKPAmg77w8oFzNFpaQiQUWD43TKaecd TokenMessengerMinter 5 CCTPiPYPc6AsJuwueEnWgSgucamXDZwBd53dQ11YiKX3 The Solana CCTP source code is available athttps://github.com/circlefin/solana-cctp-contracts/ . The interface below serves as a reference for permissionless messaging functions exposed by the programs.

Interface

The interface below serves as a reference for permissionless messaging functions exposed by the TokenMessengerMinter and MessageTransmitter programs. The full IDLs can be foundhere andhere .

Please see the instruction rust files or quick-start for PDA information.

TokenMessengerMinter

depositForBurn

Deposits and burns tokens from sender to be minted on destination domain. Minted tokens will be transferred to mintRecipient .

Parameters

Field Type Description amount u64 Amount of tokens to deposit and burn. destinationDomain u32 Destination domain identifier. mintRecipient Pubkey Public Key of token account mint recipient on destination domain.Address should be the 32 byte version of the hex address in base58. See Additional Notes on mintRecipient section for more information.

MessageSent event storage

To ensure persistent and reliable message storage, MessageSent events are stored in accounts. MessageSent event accounts are generated client-side, passed into the instruction call, and assigned to have the MessageTransmitter program as the owner. Please see theQuickstart Guide for how to generate this account and pass it to the instruction call.

For depositForBurn messages, this costs ~0.00295104 SOL in rent.This rent is paid by theevent_rent_payer account which can be the user or subsidized by a calling program or integrator.

Once an attestation is available and the message has been received on the destination chain, the event account can be closed and have the SOL reclaimed to the event_rent_payer account. This is done by calling thereclaim_event_account instruction. This can only be called by the event_rent_payer account from when the message was sent.

depositForBurnWithCaller

The same as depositForBurn but with an additional parameter, destinationCaller . This parameter specifies which address has permission to call receiveMessage on the destination domain for the message.

Parameters

Field Type Description amount u64 Amount of tokens to deposit and burn. destinationDomain u32 Destination domain identifier. mintRecipient Pubkey Public Key of mint recipient on destination domain.Address should be converted to base58. See Mint Recipient for Solana as Source Chain Transfers destinationCaller Pubkey Public Key of caller on destination domain.Address should be converted to base58.

replaceDepositForBurn

Replace a BurnMessage to change the mint recipient and/or destination caller. Allows the sender of a previous BurnMessage (created by depositForBurn or depositForBurnWithCaller) to send a new BurnMessage to replace the original. The new BurnMessage will reuse the amount and burn token of the original, without requiring a new deposit.

This is useful in situations where the user specified an incorrect address and has no way to safely mint the previously burned USDC.

Note on replaceDepositForBurn

Only the owner account of the original depositForBurn has access to call replaceDepositForBurn. The resulting mint will supersede the original mint, as long as the original mint has not confirmed yet on-chain. When using a third-party app/bridge that integrates with CCTP to burn and mint USDC, it is the choice of the app/bridge if and when to replace messages on behalf of users. When sending USDC to smart contracts, be aware of the functionality that those contracts have and their respective trust model. Parameters

Field Type Description originalMessage Vec Original message bytes (to replace). originalAttestation Vec Original attestation bytes. newDestinationCaller Pubkey The new destination caller, which may be the same as the original destination caller, a new destination caller, or an empty destination caller, indicating that any destination caller is valid.Address should be converted to base58. newMintRecipient Pubkey The new mint recipient, which may be the same as the original mint recipient, or different.Address should be converted to base58.

MessageTransmitter

receiveMessage

Messages with a given nonce can only be broadcast successfully once for a pair of domains. The message body of a valid message is passed to the specified recipient for further processing.

Parameters

Field Type Description message Vec Message bytes. attestation Vec Signed attestation of message. Remaining Accounts

If the receiveMessage instruction is being called with a deposit for burn message that will be received by the TokenMessengerMinter , additional remainingAccounts are required so they can be passed with the CPI to TokenMessengerMinter#handleReceiveMessage :

Account Name PDA Seeds PDA ProgramId isSigner? isWritable? Description token_messenger ["token_messenger"] tokenMessengerMinter false false TokenMessenger Program Account remote_token_messenger ["remote_token_messenger", sourceDomainId] tokenMessengerMinter false false Remote token messenger account where the remote token messenger address is stored for the given source domain id token_minter ["token_minter"] tokenMessengerMinter false true TokenMinter Program Account local_token ["local_token", localTokenMint.publicKey] tokenMessengerMinter false true Local token account where the information for the local token (e.g. USDCSOL) being minted is stored token_pair ["token_pair", sourceDomainId, sourceTokenInBase58] tokenMessengerMinter false false Token pair account where the info for the local and remote tokens are stored. sourceTokenInBase58 is the remote token that was burned converted into base58 format. user_token_account N/A N/A false true User token account that will receive the minted tokens. This addressmust match the mintRecipient from the source chain depositForBurn call. custody_token_account ["custody", localTokenMint.publicKey] tokenMessengerMinter false true Custody account that holds the pre-minted USDCSOL that can be minted for CCTP usage. SPL.token_program_id N/A N/A false false The native SPL token program ID. token_program_event_authority ["__event_authority"] tokenMessengerMinter false false Event authority account for the TokenMessengerMinter program. Needed to emit Anchor CPI events. program N/A N/A false false Program id for the TokenMessengerMinter program.

sendMessage

Sends a message to the destination domain and recipient. Emits a MessageSent event which will be attested by Circle’s attestation service.

Parameters

Field Type Description destinationDomain u32 Destination domain identifier. recipient Pubkey Address to handle message body on destination domain. messageBody Vec Application-specific message to be handled by recipient.

sendMessageWithCaller

Same as sendMessage but with an additional parameter, destinationCaller . This parameter specifies which address has permission to call receiveMessage on the destination domain for the message.

Parameters

Field Type Description destinationDomain u32 Destination domain identifier. recipient Pubkey Address of message recipient on destination domain. destinationCaller Pubkey Address of caller on the destination domain.Address should be converted to base58. messageBody Vec Application-specific message to be handled by recipient.

replaceMessage

Replace a message with a new message body and/or destination caller. The originalAttestation must be a valid attestation of originalMessage , produced by Circle's attestation service.

Parameters

Field Type Description originalMessage Vec Original message to replace. originalAttestation Vec Attestation of originalMessage. newMessageBody Vec New message body of replaced message. newDestinationCaller Pubkey The new destination caller, which may be the same as the original destination caller, a new destination caller, or an empty destination caller (bytes32(0) or PublicKey.default , indicating that any destination caller is valid).Address should be converted to base58.

reclaimEventAccount

Closes the given event account and reclaims the paid rent in SOL back to the event_rent_payer account. This instruction can only be called by the event_rent_payer account that paid the rent when the message was sent.

Parameters

Field Type Description attestation Vec Valid attestation for the message stored in the account. This is required to ensure the attestation service has processed and stored the message before it is deleted.

Warning

Once this instruction is executed for a message account, the message can no longer be read on-chain. We recommend not calling this instruction until the message has been received on the destination chain. If the message is lost before receiving the message, it can be fetched from the attestation service using themessages endpoint .

Additional Notes

Mint Recipient for Solana as Destination Chain Transfers

When calling depositForBurn on a non-Solana chain with Solana as the destination, the mintRecipient should be ahex encoded USDC token account address . The token accountmust exist at the time receiveMessage is called on Solana or else this instruction will revert. An example of converting an address from Base58 to hex taken from the Solana quickstart tutorial in Typescript can be seen below:

TypeScript import { bs58 } from "@coral-xyz/anchor/dist/cjs/utils/bytes"; import { hexlify } from "ethers";

const solanaAddressToHex = (solanaAddress: string): string => hexlify(bs58.decode(solanaAddress));

Mint Recipient for Solana as Source Chain Transfers

When specifying the mintRecipient for Solana depositForBurn instruction calls, the address must be given as the 32 byte version of the hex address in base58 format. An example taken fro the Solana quickstart tutorial in Typescript can be seen below:

TypeScript import { getBytes } from 'ethers'; import { PublicKey } from '@solana/web3.js';

const evmAddressToBytes32 = (address: string): string =>0x000000000000000000000000{address.replace("0x", "")};

const evmAddressToBase58PublicKey = (addressHex: string): PublicKey => new PublicKey(getBytes(evmAddressToBytes32(addressHex)));

Program Events

Program events likeDepositForBurn ,MintAndWithdraw , andMessageReceived are emitted as Anchor CPI events. This means a self-CPI is made into the program with the serialized event as instruction data so it is persisted in the transaction and can be fetched later on as needed. More information can be seen in theAnchor implementation PR , and an example of reading CPI events can be seenhere .

MessageSent events are different, as they are stored in accounts. Please see theMessageSent Event Storage section for more info. Updated6 days ago