Cross Program Invocation

In this section, we'll update the CRUD program from the previous PDA section to include Cross Program Invocations (CPIs). We'll modify the program to transfer SOL between accounts in theupdate anddelete instructions, demonstrating how to interact with other programs (in this case, the System Program) from within our program.

The purpose of this section is to walk through the process of implementing CPIs in a Solana program using the Anchor framework, building upon the PDA concepts we explored in the previous section. For more details, refer to theCross Program Invocation page.

Modify Update Instruction#

First, we'll implement a simple "pay-to-update" mechanism by modifying theUpdate struct andupdate function.

Begin by updating thelib.rs file to bring into scope items from thesystem_program module.

lib.rs use anchor_lang :: system_program :: {transfer, Transfer };

Diff

Next, update theUpdate struct to include an additional account calledvault_account . This account, controlled by our program, will receive SOL from a user when they update their message account.

lib.rs

[account(

mut , seeds = [b"vault" , user . key() . as_ref()], bump,)] pub vault_account : SystemAccount <' info

,

Diff

Explanation

Next, implement the CPI logic in theupdate instruction to transfer 0.001 SOL from the user's account to the vault account.

lib.rs let transfer_accounts = Transfer { from : ctx . accounts . user . to_account_info (), to : ctx . accounts . vault_account . to_account_info (), }; let cpi_context = CpiContext :: new (ctx . accounts . system_program . to_account_info (), transfer_accounts,); transfer (cpi_context, 1_000_000) ? ;

Diff

Explanation

Rebuild the program.

Terminal build

Modify Delete Instruction#

We'll now implement a "refund on delete" mechanism by modifying theDelete struct anddelete function.

First, update theDelete struct to include thevault_account . This allows us to transfer any SOL in the vault back to the user when they close their message account.

lib.rs

[account(

mut , seeds = [b"vault" , user . key() . as_ref()], bump,)] pub vault_account : SystemAccount <' info

, Also add thesystem_program as the CPI for the transfer requires invoking the System Program.

lib.rs pub system_program : Program <' info , System

,

Diff

Explanation

Next, implement the CPI logic in thedelete instruction to transfer SOL from the vault account back to the user's account.

lib.rs let user_key = ctx . accounts . user . key (); let signer_seeds : & [& [& [u8]]] = & [& [b"vault" , user_key . as_ref (), & [ctx . bumps . vault_account]]];

let transfer_accounts = Transfer { from : ctx . accounts . vault_account . to_account_info (), to : ctx . accounts . user . to_account_info (), }; let cpi_context = CpiContext :: new (ctx . accounts . system_program . to_account_info (), transfer_accounts,) . with_signer (signer_seeds); transfer (cpi_context, ctx . accounts . vault_account . lamports ()) ? ; Note that we updated_ctx: Context toctx: Context as we'll be using the context in the body of the function.

Diff

Explanation

Rebuild the program.

Terminal build

Redeploy Program#

After making these changes, we need to redeploy our updated program. This ensures that our modified program is available for testing. On Solana, updating a program simply requires deploying the compiled program at the same program ID.

Terminal deploy

Output

Explanation

Update Test File#

Next, we'll update ouranchor.test.ts file to include the new vault account in our instructions. This requires deriving the vault PDA and including it in our update and delete instruction calls.

Derive Vault PDA#

First, add the vault PDA derivation:

anchor.test.ts const [vaultPda , vaultBump] = PublicKey. findProgramAddressSync ([Buffer. from ("vault"), wallet.publicKey. toBuffer ()], program.programId,);

Diff

Modify Update Test#

Then, update the update instruction to include thevaultAccount .

anchor.test.ts const transactionSignature = await program.methods . update (message) . accounts ({ messageAccount: messagePda, vaultAccount: vaultPda, }) . rpc ({ commitment: "confirmed" });

Diff

Modify Delete Test#

Then, update the delete instruction to include thevaultAccount .

anchor.test.ts const transactionSignature = await program.methods . delete () . accounts ({ messageAccount: messagePda, vaultAccount: vaultPda, }) . rpc ({ commitment: "confirmed" });

Diff

Rerun Test#

After making these changes, run the tests to ensure everything is working as expected:

Terminal test

Output

You can then inspect the SolanFM links to view the transaction details, where you'll find the CPIs for the transfer instructions within the update and delete instructions.

Update CPI

Delete CPI

If you encounter any errors, you can reference thefinal code .

Next Steps#

You've completed the Solana Quickstart guide! You've learned about accounts, transactions, PDAs, CPIs, and deployed your own programs.

Visit theCore Concepts pages for more comprehensive explanations of the topics covered in this guide.

Additional learning resources can be found on theDeveloper Resources page.

Explore More Examples#

If you prefer learning by example, check out theProgram Examples Repository for a variety of example programs.

Solana Playground offers a convenient feature allowing you to import or view projects using their GitHub links. For example, open thisSolana Playground link to view the Anchor project from thisGithub repo .

Click theImport button and enter a project name to add it to your list of projects in Solana Playground. Once a project is imported, all changes are automatically saved and persisted within the Playground environment.

Previous «Program Derived Address Next Installation»