Fix Taker Private Key Handover Exploit

Name and Contact Information

- NAME: ARUNABHA DHAL
- EMAIL ID: arunabhadhal04@gmail.com
- DISCORD: Tukan003
- UNIVERSITY: INDIAN INSTITUTE OF TECHNOLOGY INDORE
- COUNTRY: INDIA
- GITHUB: https://github.com/arunabha003
- LINKEDIN: https://www.linkedin.com/in/arunabha-dhal-23189624b/

Synopsis:

This project addresses a vulnerability in the Coinswap protocol where a taker can unfairly complete a swap by withholding their private key from earlier makers, forcing honest makers to use on-chain fallback transactions that increase fees and compromise privacy. The solution is to reverse the key handover order: after the hash preimage is revealed, the taker must first send their private key to the maker, who verifies its validity before sending their own key. This change ensures fairness, prevents exploitation, and preserves the privacy and efficiency of the Coinswap protocol.

Project Plan:

1. Introduction and Motivation

In the current Coinswap protocol flow, there exists a critical vulnerability that undermines the fairness of the system. After analyzing the protocol flow diagram and codebase, we've identified that the taker (Alice) can complete a coinswap by only revealing the hash preimage to the last maker in the hop chain (Charlie), thereby obtaining the multisig private keys for that final swap. However, Alice can then refuse to share her own multisig private key with earlier makers like Bob.

This asymmetry creates a situation where:

- The taker receives the benefits of the swap (obtaining Charlie's private key).
- Earlier makers are forced to use on-chain contract transactions (HTLCs) to claim their funds
- These on-chain fallbacks incur additional mining fees

- Privacy is compromised by revealing the swap link on-chain.
- Honest makers are unfairly punished despite successful swap completion

The vulnerability exists because of the sequence in which keys are exchanged. Currently, after receiving the hash preimage, makers send their private keys first, giving takers the ability to complete their portion of the swap without fulfilling obligations to earlier participants.

---

2. Reversed Key Handover Solution

Goal: Ensure protocol fairness by modifying the private key handover sequence so takers must fulfill their obligations before receiving the maker's private key.

Implementation Plan: We will modify the key handoff sequence in the protocol as follows:

1. Message Flow Revision:

Rather than having the maker send their private key first after receiving the hash preimage, we will reverse the order:

- Taker sends hash preimage to maker
- Taker sends their private key to maker
- Maker verifies the private key is valid
- Only after successful verification does maker send their private key to taker

This ensures that takers cannot gain an unfair advantage without first providing their portion of the multisig.

2. Protocol Message Types:

We will replace the generic `RespPrivKeyHandover` message with specific message types to clearly distinguish the direction of key handover:

- `TakerPrivKeyHandover` (Taker → Maker)
- `MakerPrivKeyHandover` (Maker → Taker)

This distinction ensures clear separation of responsibilities and makes the protocol flow explicit.

3. Key Verification Process:

When the maker receives the taker's private key, they must verify:

- The private key is valid (properly formatted)
- The key corresponds to the expected public key previously shared
- The key can correctly derive the multisig address used in the funding transaction

Only if all verifications pass will the maker send their own private key. This prevents takers from sending

invalid keys.

4. Fallback Mechanism:

If verification fails or timeouts occur:

- The maker will not send their private key
- Both parties will fall back to on-chain HTLC contract transactions
- The hash preimage will still allow claiming funds on-chain
- An appropriate error message will be returned to the taker

This ensures protocol robustness even when a party attempts to cheat.

---

3. Code Implementation Details:

1. Message Protocol Updates (src/protocol/messages.rs):

This code modifies the protocol message types to support directional private key handover. We're replacing the generic **RespPrivKeyHandover** message with two distinct messages:

- **TakerPrivKeyHandover**: Sent from taker to maker containing the taker's private key
- **MakerPrivKeyHandover**: Sent from maker to taker containing the maker's private key

This directional distinction is crucial for enforcing the correct order of key exchange. We also add a new error type **InvalidPrivateKey** to the **ErrorPayload** enum which will be used when the maker detects that the taker has provided an invalid key.

Code Template :

```
// BEFORE
pub enum Message {
    RespPrivKeyHandover(PrivateKeyData),
    // ... other messages ...
}

// AFTER
pub enum Message {
    TakerPrivKeyHandover(PrivateKeyData),  // New directional message
    MakerPrivKeyHandover(PrivateKeyData),  // New directional message
    // ... other messages ...
}
```

2. Key Verification Implementation (src/protocol/verification.rs):

This code implements a new **KeyVerifier** class that handles the critical security verification of the taker's private key. The verification happens in two parts:

1. **verify_public_key_match**: This method confirms that the provided private key actually corresponds to the public key that the taker previously shared during the protocol handshake. This ensures the taker isn't providing a different key.
2. **verify_multisig_capability**: This method checks that the private key, when combined with the maker's public key, correctly recreates the multisig address used in the funding transaction. This verifies that the key can actually be used to spend from the multisig address.
3. **create_multisig_address:** This helper function implements the Bitcoin-specific logic for creating a 2-of-2 multisig address from two public keys. This follows the same pattern used elsewhere in the codebase to ensure consistent address generation.

Code Template :

```
// src/protocol/verification.rs
pub struct KeyVerifier {
    // expected_public_key, multisig_address, secp context...
}

impl KeyVerifier {
    pub fn new(expected_pk: PublicKey, ms_addr: Address) -> Self { /*...*/ }
    pub fn verify_public_key_match(&self, sk: &SecretKey) -> bool { /*...*/ }
    pub fn verify_multisig_capability(&self, sk: &SecretKey, mpk: &PublicKey) -> bool { /*...*/ }
    pub fn verify(&self, sk: &SecretKey, mpk: &PublicKey) -> bool {
        self.verify_public_key_match(sk) && self.verify_multisig_capability(sk, mpk)
    }
}

fn create_multisig_address(k1: &PublicKey, k2: &PublicKey) -> Address { /*...*/ }
```

3. Taker Protocol Modifications (src/taker/protocol.rs):

This code modifies the taker's protocol implementation to use the new key handover sequence. The key changes are:

1. **send_private_key**: Now uses the specific TakerPrivKeyHandover message type instead of the generic one, making the protocol flow explicit.
2. **receive_maker_private_key**: Modified to expect the specific MakerPrivKeyHandover message type from the maker.
3. **complete_swap:** The most important change - this method implements the new sequence where the taker sends their private key first, then waits for the maker's key. This is the reverse of the original sequence and prevents the exploit.

Code Template :

```
// src/taker/protocol.rs
impl TakerProtocol {
    pub fn send_hash_preimage(&self, x: HashPreimage) -> Result<(), ProtocolError> { /*...*/ }
    pub fn send_private_key(&self, key: PrivateKeyData) -> Result<(), ProtocolError> { /*...*/
}
    pub fn receive_maker_private_key(&self) -> Result<PrivateKeyData, ProtocolError> { /*...*/
}

    pub fn complete_swap(&self, hop_index: usize) -> Result<(), ProtocolError> {
        // 1) send_hash_preimage
        // 2) send_private_key
        // 3) receive_maker_private_key
        Ok(())
    }
}
```

4. Maker Protocol Modifications (src/maker/protocol.rs):

This code modifies the maker's protocol implementation to:

1. **receive_taker_private_key**: Expect and process the new `TakerPrivKeyHandover` message type from the taker.
2. **send_private_key:** Use the new `MakerPrivKeyHandover` message type when sending the maker's private key to the taker.
3. **handle_private_key_handover:** This is the critical method that implements the new secure flow:
    - First, receive and verify the hash preimage
    - Then, receive the taker's private key
    - Verify that the taker's key is valid for the multisig address
    - Only after successful verification, send the maker's private key
    - If verification fails, send an error message and do not reveal the maker's key

The verification step is crucial - it uses the `KeyVerifier` class we defined earlier to ensure that the taker has provided a valid private key that corresponds to their public key and can be used for the multisig address.

Code Template :

```
// src/maker/protocol.rs
impl MakerProtocol {
    pub fn receive_hash_preimage(&self) -> Result<HashPreimage, ProtocolError> { /*...*/ }
    pub fn receive_taker_private_key(&self) -> Result<PrivateKeyData, ProtocolError> { /*...*/
}
    pub fn send_private_key(&self, key: PrivateKeyData) -> Result<(), ProtocolError> { /*...*/
}

    pub fn handle_private_key_handover(&self) -> Result<(), ProtocolError> {
        // receive_hash_preimage → verify → receive_taker_private_key → verify →
```

```
send_private_key
        Ok(())
    }
}
```

---

4. Testing Strategy

Our testing approach will validate both the cryptographic correctness of the key verification and the protocol flow changes:

1. Unit Tests:

1.Private Key Verification Tests:

These unit tests validate the core cryptographic verification functionality:

- **test_valid_private_key_verification**: Tests that a legitimate private key is correctly verified. This ensures that our verification logic accepts valid keys that match the expected public key and can be used for the multisig address.
- **test_invalid_private_key_verification**: Tests that a wrong private key is rejected. This ensures that our verification logic correctly detects when an invalid key is provided that doesn't match the expected public key.
- **test_public_key_match_only:** Tests a specific edge case where a key matches the public key but isn't valid for the multisig address when paired with a different maker key. This ensures our verification is comprehensive and checks both aspects correctly.

These tests use the Bitcoin secp256k1 library to generate actual cryptographic keys and test the verification logic with real cryptographic operations, ensuring that our implementation is cryptographically sound.

Code Template :

```
// tests/unit/test_private_key_verification.rs
#[cfg(test)]
mod tests {
    use bitcoin::secp256k1::{Secp256k1, SecretKey, PublicKey};
    use crate::protocol::verification::KeyVerifier;

    #[test]
    fn test_valid_private_key_verification() {
        // setup sk/pk pair, multisig addr...
        // let verifier = KeyVerifier::new(pk, ms_addr);
        // assert!(verifier.verify(&sk, &maker_pk));
    }
```

```
    #[test]
    fn test_invalid_private_key_verification() {
        // setup correct pk but wrong sk...
        // assert!(!verifier.verify(&wrong_sk, &maker_pk));
    }

    #[test]
    fn test_public_key_match_only() {
        // sk matches pk but not multisig with different maker_pk...
        // assert!(!verifier.verify_multisig_capability(&sk, &diff_maker_pk));
    }
}
```

2. Protocol Message Tests:

These tests validate the protocol message serialization and deserialization:

- **`test_taker_privkey_handover_message_serialization`:** Tests that the new `TakerPrivKeyHandover` message type can be correctly serialized to bytes and then deserialized back to the original message. This ensures that our message protocol changes are correctly implemented and can be used for network communication.
- **`test_maker_privkey_handover_message_serialization`:** Similarly tests the new `MakerPrivKeyHandover` message type, ensuring bidirectional message handling works correctly.
- **`test_invalid_private_key_error_serialization`:** Tests the new `InvalidPrivateKey` error type, ensuring that error messages can be correctly transmitted when verification fails.

These tests ensure that the protocol changes we've made will work correctly at the network communication level, preventing issues when the updated protocol is deployed.

Code Template:

```
// tests/unit/test_protocol_messages.rs
#[cfg(test)]
mod tests {
    use crate::protocol::messages::{Message, ErrorPayload, PrivateKeyData};

    #[test]
    fn test_taker_privkey_handover_serialization() {
        // let data = PrivateKeyData::new_random();
        // let msg  = Message::TakerPrivKeyHandover(data.clone());
        // round-trip serialize/deserialize and assert_eq!(data, decoded);
    }

    #[test]
    fn test_maker_privkey_handover_serialization() {
        // same as above for MakerPrivKeyHandover
    }

    #[test]
```

```
    fn test_invalid_private_key_error_serialization() {
        // let msg = Message::Error(ErrorPayload::InvalidPrivateKey);
        // serialize/deserialize → match ErrorPayload::InvalidPrivateKey
    }
}
```

3. Integration Tests:

These integration tests validate the complete protocol flow with the new key handover sequence:

1. **`test_successful_coinswap_with_new_key_handover`:** Tests the successful case where all parties follow the protocol correctly. It sets up a complete multi-hop swap and verifies that:
    a. The swap completes successfully
    b. All parties receive the expected funds
    c. The swap is completed off-chain (no contract transactions needed)
2. **`test_taker_provides_invalid_key`:** Tests what happens when a taker tries to provide an invalid private key. It verifies that:
    a. The maker correctly detects the invalid key
    b. The protocol fails with the appropriate error
    c. The maker falls back to the contract transaction
    d. The maker can still claim their funds via the contract
3. **`test_multi_hop_swap_with_malicious_taker`:** Tests a more complex scenario where a taker is honest with one maker but malicious with another. This tests that:
    a. The honest maker receives their funds off-chain
    b. The maker who received an invalid key falls back to the contract transaction.
    c. Both makers can still claim their funds, just through different mechanisms

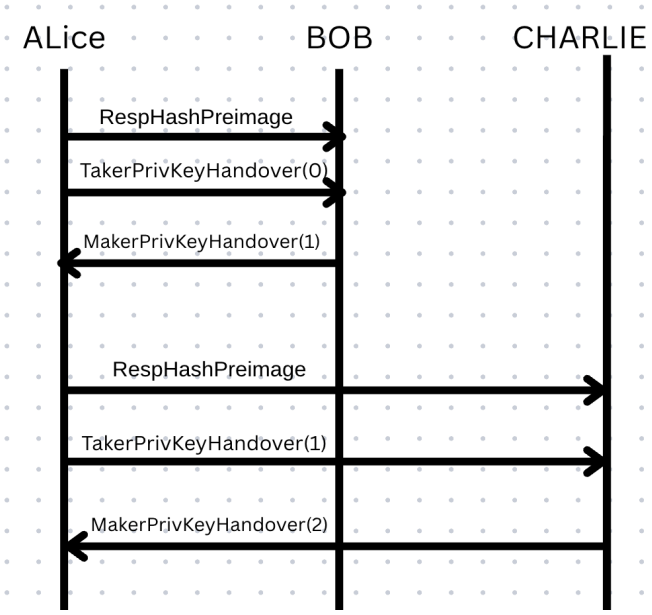
Code Template:

```
// tests/integration/test_standard_coinswap.rs
#[cfg(test)]
#[cfg(feature = "integration-test")]
mod tests {
    use crate::test_utils::{TestEnvironment};

    #[test]
    fn test_successful_coinswap_with_new_key_handover() {
        // setup regtest env, nodes A/B/C...
        // env.execute_swap_with_hops(&[&bob, &charlie])?;
        // assert!(alice.has_received_expected_funds());
    }

    #[test]
    fn test_taker_provides_invalid_key() {
        // configure alice to send bad key...
        // assert!(alice.execute_swap().is_err());
        // assert!(bob.has_used_contract_transaction());
```

```
    }

    #[test]
    fn test_multi_hop_swap_with_malicious_taker() {
        // alice honest to C, malicious to B...
        // let res = alice.execute_swap();
        // assert!(res.is_ok());
        // assert!(bob.used_contract());
        // assert!(!charlie.used_contract());
    }
}
```

---

5. Flow Diagrams



---

6. Timeline:

Weeks 0-1 (Community Bonding):

- Finalize technical design with mentor
- Explore existing Coinswap protocol implementation
- Set up local development environment with Bitcoin regtest node

Weeks 1-2 (Protocol Message Updates):

- Update message types in protocol to support directional key handover.
- Implement error handling for invalid private keys
- Create unit tests for message serialization/deserialization

Weeks 3-4 (Key Verification Implementation):

- Implement the private key verification module
- Create unit tests for key verification
- Integrate verification into the protocol flow

Weeks 5-6 (Taker Protocol Updates):

- Modify taker protocol to send private key first
- Update taker's handling of key exchange sequence
- Create integration tests for taker protocol

Week 6 (Midterm Evaluation):

- Demonstrate working taker-side implementation
- Review progress with mentor
- Plan adjustments for second half

Weeks 7-8 (Maker Protocol Updates):

- Modify maker protocol to verify keys before responding
- Update maker's key handover sequence
- Create integration tests for maker protocol

Weeks 9-10 (End-to-End Testing):

- Implement full integration tests for multi-hop swaps
- Test adversarial scenarios (invalid keys, timeouts)
- Fix edge cases and improve error handling

Week 11 (Documentation and Diagrams):

- Create detailed flow diagrams for the protocol
- Write comprehensive documentation
- Prepare final demonstration

Week 12 (Final Submission):

- Submit final code with all tests passing
- Complete project documentation
- Present solution to Coinswap team

Future Deliverables :

I want to work on the issues for v0.1.2. Also want to get done with the issue “Fee rate negotiation” as I’m already working on it.

Benefits to Community :

This project enhances the security and fairness of the Coinswap protocol, ensuring that all participants are protected from exploitation and that honest makers are not penalized by unnecessary fees or privacy loss. By closing this vulnerability, the protocol becomes more trustworthy and attractive for both makers and takers, encouraging greater adoption and participation. Ultimately, this strengthens Bitcoin's privacy ecosystem, empowering users with more robust, reliable tools for confidential transactions.

Biographical Information :

I'm a 3rd-year undergrad at IIT Indore, and I've been actively working in the EVM ecosystem as a Solidity developer and security researcher, with backend experience in JavaScript and Python, plus DevOps. I've spent a lot of time in DeFi on EVM, and I've always wanted to explore its scope on the Bitcoin network. In the past, I've worked with Ordinals and Runes, and I'm currently involved in a Bitcoin–TON bridging project.

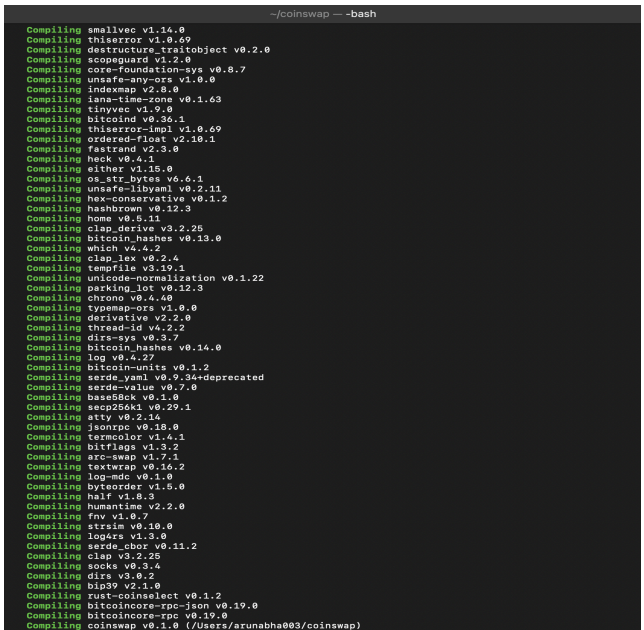
When I looked into past SOB organizations, Coinswap immediately caught my eye. After running the demo, I was hooked—I dove into the codebase and decided to contribute. I believe I'm an ideal candidate because I'm genuinely committed to getting Coinswap production-ready. I've worked with Move (a Rust-like language) before.Still I consider myself as a rust beginner, and on a continuous process of learning more. Working across different domains allows me to pick up a new tech stack, learn it, and implement it very quickly. When I'm stuck on a problem, I work for hours straight until I get it solved—something I think really sets me apart.

To date, I've had two PRs merged into the Coinswap codebase and another PR is up for review.
1.https://github.com/citadel-tech/coinswap/pull/455 (merged)
2.https://github.com/citadel-tech/coinswap/pull/452 (merged)

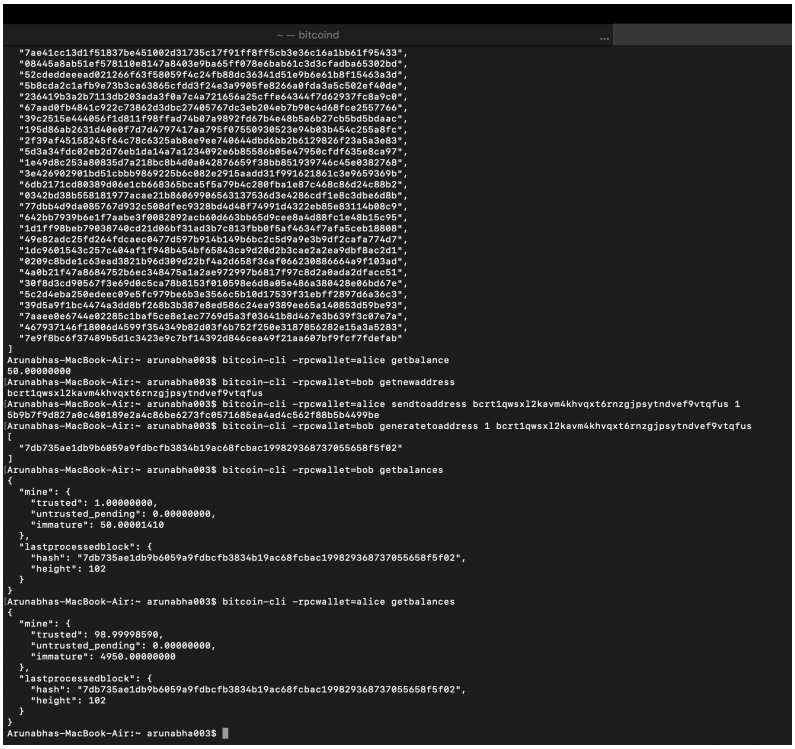
3.https://github.com/citadel-tech/coinswap/pull/486 (open PR)

Competency Test:

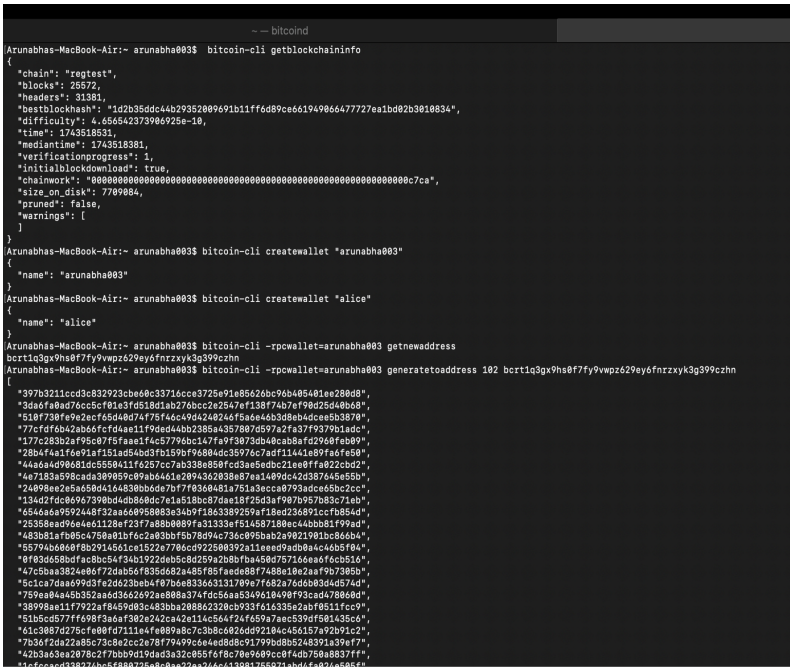
1.Compiling the Coinswap project, and running all tests.



2. Setting up a local Bitcoin Core node in regtest mode. Creating a wallet, receiving funds, and sending transactions using bitcoin-cli.

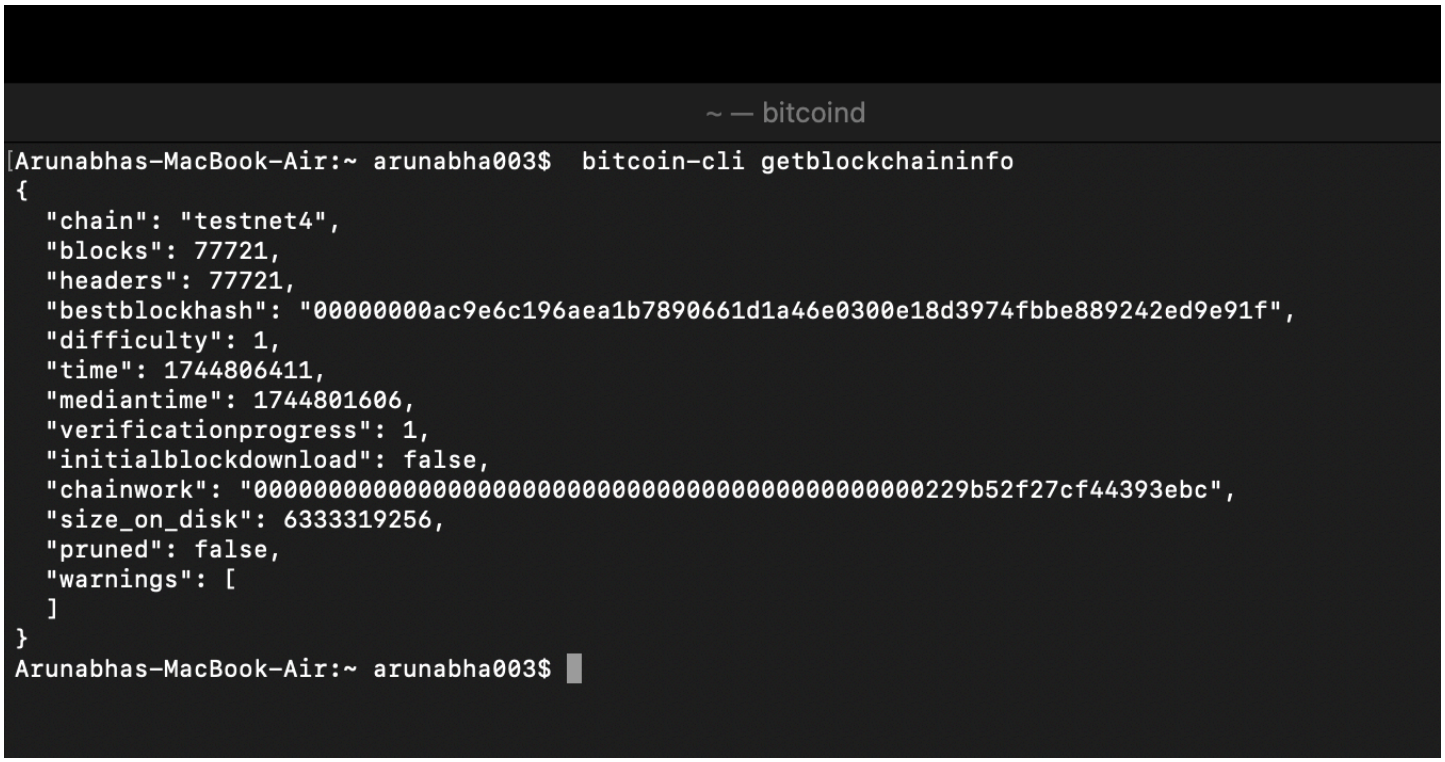


```
~ — bitcoind
  "7ae41cc13d1f51837be451002d31735c17f91ff8ff5cb3e36c16a1bb61f95433",
  "08445a8ab51ef578110e8147a8403e9ba65ff078e6bab61c3d3cfadba65302bd",
  "52cdeddeeeead021266f63f58059f4c24fb88dc36341d51e9b6e61b8f15463a3d",
  "5b8cda2c1afb9e73b3ca63865cfdd3f24e3a9905fe8266a0fda3a8c502ef40de",
  "236419b3a2b7133db203ada3f0a7c4a721656a2bcffe6434ef7d62937fc8a9c8",
  "67aad0fb4841c922c73862d3dbc27405767dc3eb204eb7b90c4d68fce2557766",
  "39c2515e444056f1d811f98ffad74b07a9892fd67b4e48b5a6b27cb5bd5bdaac",
  "195d86ab2631d40e0f7d7d4797417aa795f07550930523e94b03b454c255a8fc",
  "3f39af4515825f64c78c6325abb8e09ee740644dbd6bb2b6129826f23a5a3e03",
  "5d3a34fdc02eb2d76eb1da14a7a1234092e6b85586b05e47950cfdf635e8ca97",
  "1e49d8c253a80835d7a218bc8b4d0a042876659f38bb851939746c45e0382768",
  "3e426902901bd51cbbb9869225bb6c082e2915aadd31f991621861c3e9659369b",
  "4db2171cd80389d06e1cb668365bca5f5a79b4c280fba1e87c468c86d24c88b2",
  "8342bd38b558181977acae21b8606990565137536d3e4286cdf1e8c3dbc6d8b",
  "77dbb4d9da085767d932c508dfec9328bd4d48f74991d4322eb856e83114b08c9",
  "642bb7939b6e1f7aabe3f0082892acb60d663bb65d9cee8a4d88fc1e48b15c95",
  "1d1ff98beb79038740cd21d06bf31ad3b7c813fbb0f5af4634f7afa5ceb18808",
  "49e82adc25fd264fdcaec0477d597b914b149b6bc2c5d9a9e3b9df2cafa774d7",
  "1dc9601543c257c40aaf1f94bb454bf65843ca9d20d2b3cae2a2ea9dbf8ac2d1",
  "0209c8bde1c63ead3821b96d309d22bf4a2d658f36af06623088666a9f103ad",
  "4a0b21f47a8684752b6ec348475a1a2ae972997b6817f97c8d2a0ada2dfacc51",
  "30f8d3cd09567f3e69d0c5ca78b8153f010598e6d8a05e486a380428e06bb67e",
  "5c2d4eba250edeec09e5fc970be6b3e3566c5b10d17539f31ebff2897d6a36c3",
  "39d5a9f1bc4474a3dd8bf268b3b387e8ed586c24ea9389ee65a148853d59be93",
  "7aaee0e6744e02285c1baf5ce8e1ec7769d5a3f03641b8d467e3b639f3c07e7a",
  "467937146f18006d4599f354349b82d03f6b752f250e3187856282e15a3a5283",
  "7e9f8bc6f37489b5d1c3423e9c7bf14392d846cea49f21ae607bf9fcf7fdefab"
]
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli -rpcwallet=alice getbalance
50.00000000
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli -rpcwallet=bob getnewaddress
bcrt1qwsxl2kavm4khvqxt6rnzgjpsytndvef9vtqfus
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli -rpcwallet=alice sendtoaddress bcrt1qwsxl2kavm4khvqxt6rnzgjpsytndvef9vtqfus 1
5b9b7f9d827a0c480189e2a4c86be6273fc0571685ea4ad4c562f88b5b4499be
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli -rpcwallet=bob generatetoaddress 1 bcrt1qwsxl2kavm4khvqxt6rnzgjpsytndvef9vtqfus
[
  "7db735ae1db9b6059a9fdbcfb3834b19ac68fcbac19982936873705 5658f5f02"
]
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli -rpcwallet=bob getbalances
{
  "mine": {
    "trusted": 1.00000000,
    "untrusted_pending": 0.00000000,
    "immature": 50.00001410
  },
  "lastprocessedblock": {
    "hash": "7db735ae1db9b6059a9fdbcfb3834b19ac68fcbac19982936873705 5658f5f02",
    "height": 102
  }
}
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli -rpcwallet=alice getbalances
{
  "mine": {
    "trusted": 98.99998590,
    "untrusted_pending": 0.00000000,
    "immature": 4950.00000000
  },
  "lastprocessedblock": {
    "hash": "7db735ae1db9b6059a9fdbcfb3834b19ac68fcbac19982936873705 5658f5f02",
    "height": 102
  }
}
Arunabhas-MacBook-Air:~ arunabha003$
```



```
~ — bitcoind
Arunabhas-MacBook-Air:~ arunabha003$  bitcoin-cli getblockchaininfo
{
  "chain": "regtest",
  "blocks": 25572,
  "headers": 31381,
  "bestblockhash": "1d2b35ddc44b29352009691b11ff6d89ce661949066477727ea1bd02b3010834",
  "difficulty": 4.656542373906925e-10,
  "time": 1743518531,
  "mediantime": 1743518381,
  "verificationprogress": 1,
  "initialblockdownload": true,
  "chainwork": "0000000000000000000000000000000000000000000000000000000000000c7a",
  "size_on_disk": 7709084,
  "pruned": false,
  "warnings": [
  ]
}
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli createwallet "arunabha003"
{
  "name": "arunabha003"
}
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli createwallet "alice"
{
  "name": "alice"
}
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli -rpcwallet=arunabha003 getnewaddress
bcrt1q3gx9hs0f7fy9vwpz629ey6fnrzxyk3g399czhn
Arunabhas-MacBook-Air:~ arunabha003$ bitcoin-cli -rpcwallet=arunabha003 generatetoaddress 102 bcrt1q3gx9hs0f7fy9vwpz629ey6fnrzxyk3g399czhn
[
  "397b3211ccd3c832923cbe60c33716cce3725e91e85626bc96b405401ee280d8",
  "3da6fa0ad76cc5cf01e3fd518d1ab276bcc2e2547ef138f74b7ef90d25d40b68",
  "510f730fe9e2ecf65d40d74f75f46c49d4240246f5a6e46b3d8eb4dcee5b3870",
  "77cfdf6b42ab66fcfd4ae11f9ded44bb2385a4357807d597a2fa3f9379b1adc",
  "177c283b2af95e07f5faae1f4c5779bc147fa9f3073db40cab8afd296ffab09",
  "28b4f4a1f6e91af151ad54bd3fb159bf96804dc35976c7adf11441e89fa6fe50",
  "44a6a4d90681dc5550411f6257cc7ab338e850fcd3ae5edbc21ee0ffa022cbd2",
  "4e7183a598cada309059c09ab6461e2094362038e87ea1409dc42d387645e55b",
  "24098ee2e5a650d416480 0bb6de7bf7f0360481a751a3ecca0793adce65bc2cc",
  "134d2fdc8696739 0bd4db860dc7e1a518bc87dae18f25d3af907b957b83c71eb",
  "454a6a9592448f32aa66 958083e34b9f186338925 9af18ed236891ccfb854d",
  "25358ead96e4e61128ef23f7a88b0089fa31333ef511587180ec44bbb81f99ad",
  "483b81afb05c4750e01bf6c2a03bbf5b78d94c736c095bab2a9021901bc866b4",
  "55794b0060f8b2914561ce1522e7706cd922500392a11eeed9adb0a4c46b5f04",
  "0f03d658bdfac8bc54f34b1922deb5c8d259a2b8bfba450d757166ea6f6cb516",
  "47c5baa3824e06f72dab56f835d682a485f85faede88f7488e10e2aaf9b7305b",
  "5c1ca7daa699d3fe2d623beb4f07b6e8336 3131709e7f682a76d6b03d4d574d",
  "759ea84a45b352aa6d3662692ae808a374fdc56aa5349610498f93cad47860d0d",
  "38998ae11f7922af8459d03c483bba2088 2320cb933f616335e2abf0511fcc9",
  "51b5cd577ff698f3a6af302e242ca42e114c564f24f659a7aec539df501435c6",
  "61c3087d275cfe00fd7111e4fe089a8c7c3b8c6026dd9210c4c456157a92b91c2",
  "7b36f2da22a85c73c8e2cc2e78f79499c6e4ed8d8c91799bd8b5248391a39ef7",
  "42b3a63ea2078c2f7bbb9d19dad3a32c055f6f8c70e9609cc0f4db750a883 7ff",
```

3. Syncing a Testnet4 node locally. (as of 16th April, 16:04)



```
~ — bitcoind
[Arunabhas-MacBook-Air:~ arunabha003$  bitcoin-cli getblockchaininfo
{
  "chain": "testnet4",
  "blocks": 77721,
  "headers": 77721,
  "bestblockhash": "00000000ac9e6c196aea1b7890661d1a46e0300e18d3974fbbe889242ed9e91f",
  "difficulty": 1,
  "time": 1744806411,
  "mediantime": 1744801606,
  "verificationprogress": 1,
  "initialblockdownload": false,
  "chainwork": "00000000000000000000000000000000000000000000229b52f27cf44393ebc",
  "size_on_disk": 6333319256,
  "pruned": false,
  "warnings": [
  ]
}
Arunabhas-MacBook-Air:~ arunabha003$
```

4.Successful Coinswap

```
2025-04-10T21:06:43.891350+05:30 INFO coinswap::taker::api - Connecting to ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202 | Send Sigs Init Next Hop
2025-04-10T21:06:45.589807+05:30 INFO coinswap::taker::api - ===> ProofOfFunding | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:06:45.589849+05:30 INFO coinswap::taker::api - Fundix Txids: [79eedbf3a0db59fcdca8563acf67b01df9c66f28fe930e683bea22399f71f211]
2025-04-10T21:06:47.169611+05:30 INFO coinswap::taker::routines - Maker Received = 0.00020000 BTC | Maker is Forwarding = 0.00019540 BTC |  Coinswap Fees = 0.00000160 BTC  | Miner Fees paid by us = 300
2025-04-10T21:06:47.169714+05:30 INFO coinswap::taker::api - <=== ReqContractSigsAsRecvrAndSender | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:06:47.791852+05:30 INFO coinswap::taker::api - ===> ReqContractSigsForSender | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:06:50.834898+05:30 INFO coinswap::taker::api - <=== RespContractSigsForSender | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:06:50.835057+05:30 INFO coinswap::taker::api - Taker is previous peer. Signing Receivers Contract Txs
2025-04-10T21:06:50.835435+05:30 INFO coinswap::taker::api - ===> RespContractSigsForRecvrAndSender | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:06:50.835541+05:30 INFO coinswap::taker::api - Waiting for funding transaction confirmation. Txids : [a98726196daef4225ca0588e4b31924999e7eee0eb71c9c2b97503c9c514690d]
2025-04-10T21:06:50.837215+05:30 INFO coinswap::taker::api - Waiting for funding tx to appear in mempool | 0 secs
2025-04-10T21:07:20.843892+05:30 INFO coinswap::taker::api - Funding tx Seen in Mempool. Waiting for confirmation for 30 secs
2025-04-10T21:07:21.527506+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:07:22.057544+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:07:52.065813+05:30 INFO coinswap::taker::api - Funding tx Seen in Mempool. Waiting for confirmation for 61 secs
2025-04-10T21:07:52.478080+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:07:52.937897+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:08:22.947142+05:30 INFO coinswap::taker::api - Funding tx Seen in Mempool. Waiting for confirmation for 92 secs
2025-04-10T21:08:23.518768+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:08:24.132959+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:08:54.142618+05:30 INFO coinswap::taker::api - Tx a98726196daef4225ca0588e4b31924999e7eee0eb71c9c2b97503c9c514690d | Confirmed at 1
2025-04-10T21:08:54.143370+05:30 INFO coinswap::taker::api - Connecting to m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302 | Send Sigs Init Next Hop
2025-04-10T21:08:55.147991+05:30 INFO coinswap::taker::api - ===> ProofOfFunding | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:08:55.148087+05:30 INFO coinswap::taker::api - Fundix Txids: [a98726196daef4225ca0588e4b31924999e7eee0eb71c9c2b97503c9c514690d]
2025-04-10T21:08:56.948155+05:30 INFO coinswap::taker::routines - Maker Received = 0.00019540 BTC | Maker is Forwarding = 0.00019100 BTC |  Coinswap Fees = 0.00000140 BTC  | Miner Fees paid by us = 300
2025-04-10T21:08:56.948282+05:30 INFO coinswap::taker::api - <=== ReqContractSigsAsRecvrAndSender | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:08:56.948376+05:30 INFO coinswap::taker::api - Taker is next peer. Signing Sender's Contract Txs
2025-04-10T21:08:57.617763+05:30 INFO coinswap::taker::api - ===> ReqContractSigsForRecvr | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:09:02.544088+05:30 INFO coinswap::taker::api - <=== RespContractSigsForRecvr | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:09:02.544193+05:30 INFO coinswap::taker::api - ===> RespContractSigsForRecvrAndSender | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:09:02.544287+05:30 INFO coinswap::taker::api - Waiting for funding transaction confirmation. Txids : [81e3faabcd3384d2cb59e02f2427a9504fd8f7f5b6b131561ff0e07bce4e014b]
2025-04-10T21:09:02.545635+05:30 INFO coinswap::taker::api - Waiting for funding tx to appear in mempool | 0 secs
2025-04-10T21:09:32.551022+05:30 INFO coinswap::taker::api - Funding tx Seen in Mempool. Waiting for confirmation for 30 secs
2025-04-10T21:09:33.256850+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:09:33.764786+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:10:03.772424+05:30 INFO coinswap::taker::api - Funding tx Seen in Mempool. Waiting for confirmation for 61 secs
2025-04-10T21:10:04.382395+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:10:04.880774+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:10:34.890545+05:30 INFO coinswap::taker::api - Funding tx Seen in Mempool. Waiting for confirmation for 92 secs
2025-04-10T21:10:35.466378+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:10:36.342316+05:30 INFO coinswap::taker::api - ===> WaitingFundingConfirmation | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:11:06.352013+05:30 INFO coinswap::taker::api - Tx 81e3faabcd3384d2cb59e02f2427a9504fd8f7f5b6b131561ff0e07bce4e014b | Confirmed at 1
2025-04-10T21:11:07.365458+05:30 INFO coinswap::taker::api - ===> ReqContractSigsForRecvr | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:11:10.672015+05:30 INFO coinswap::taker::api - <=== RespContractSigsForRecvr | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:11:13.097064+05:30 INFO coinswap::taker::api - ===> HashPreimage | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:11:13.658045+05:30 INFO coinswap::taker::api - <=== PrivateKeyHandover | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:11:13.658615+05:30 INFO coinswap::taker::api - ===> PrivateKeyHandover | ewaexd2es2uzr34wp26cj5zgph7bug7znmmxolvwzmoeedbiyfgz3wqd.onion:8202
2025-04-10T21:11:14.834138+05:30 INFO coinswap::taker::api - ===> HashPreimage | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:11:15.448214+05:30 INFO coinswap::taker::api - <=== PrivateKeyHandover | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:11:15.448554+05:30 INFO coinswap::taker::api - ===> PrivateKeyHandover | m3qn53qt3wukwqvbx4yhq7v2qrsvf2oiddtjsqigwktzuuot33hmmuad.onion:8302
2025-04-10T21:11:15.448713+05:30 INFO coinswap::taker::api - Initializing Sync and Save.
2025-04-10T21:11:15.803433+05:30 INFO coinswap::taker::api - Completed Sync and Save.
2025-04-10T21:11:15.803439+05:30 INFO coinswap::taker::api - Successfully Completed Coinswap.
2025-04-10T21:11:15.803446+05:30 INFO coinswap::taker::api - Shutting down taker.
2025-04-10T21:11:15.803484+05:30 INFO coinswap::taker::api - offerbook data saved to disk.
2025-04-10T21:11:15.803522+05:30 INFO coinswap::taker::api - Wallet data saved to disk.
Arunabhas-MacBook-Air:coinswap arunabha003$
```