Wallet Encryption and Backup Mechanism

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 aims to implement robust wallet encryption and backup features for the Coinswap protocol, addressing critical security and usability gaps. Currently, Coinswap wallets are stored in plaintext, making them vulnerable to compromise. Additionally, the absence of automated backups puts users at risk of losing funds due to file corruption or disk failure.

To solve this, the project introduces AES-256-GCM encryption with PBKDF2-HMAC-SHA256-based password protection for both maker and taker wallets. It also adds local encrypted backups triggered automatically on shutdown and manually via CLI. Optional cloud backups to a versioned AWS S3 bucket will provide off-site recovery, with setup automated via Terraform. CLI commands for backup and restore will be integrated into `maker-cli` and `taker`, alongside secure password prompts and overwrite safeguards.

Project Plan:

1. Introduction and Motivation

This project proposes to integrate strong, password-based encryption for both maker and taker wallets, along with secure local and optional cloud backup features. The goal is to ensure that wallets remain private and recoverable, even in adverse scenarios—strengthening both the security and usability of the Coinswap ecosystem.

---

2. AES-256-GCM Wallet Encryption (PBKDF2-HMAC-SHA256)

Goal: Ensure only someone with the correct password can read the wallet file. We'll derive a 256-bit key from a user password and salt (using PBKDF2-HMAC-SHA256), then encrypt the wallet data with AES-256-GCM.

Implementation Plan: We will integrate encryption into the wallet creation and loading process as follows:

1.Password Setup:
During initial wallet creation, prompt the user (or read from config) for a wallet password. For `makerd`, this occurs on startup if no encrypted wallet exists; for `taker`, on first CLI use.If no password is provided(empty string), wallet data remains unencrypted for backward compatibility. The password is **not** stored in plaintext anywhere; it's only used to derive the encryption key.

2.Key Derivation:

- Generate a 16-byte random salt for each wallet and store that salt alongside the encrypted wallet data on disk. .
- We will use PBKDF2-HMAC-SHA256 with a high iteration count (100k iterations of SHA256 is fast enough to keep wallet load under a couple of seconds) to derive the 256-bit key from the user's password and salt.
- The derived key is used for AES-256-GCM encryption/decryption.

3.Encryption on Save:

- Whenever the wallet is serialized to disk (e.g. on shutdown or state save),*the wallet plaintext is then encrypted with AES-GCM using a* 12-byte IV (nonce)*, producing ciphertext and an authentication tag, which are stored to the wallet file.*
- The output file format will contain a simple header (magic bytes like `COINSWAP` for identification), the salt, the IV, and the ciphertext+tag. For example, the file layout might be: `[ MAGIC | salt(16) | iv(12) | ciphertext+tag]`

4.Decryption on Load:

- When loading the wallet (at `makerd` startup or when the taker cli used), require the user's password to decrypt.
- The code will read the header to detect if the wallet is encrypted.If so, it reads the salt and IV from the file, runs PBKDF2 to regenerate the key, then decrypts the ciphertext using AES-256-GCM.
- We will prompt for password, run PBKDF2, decrypt via AES-256-GCM. If the password is wrong, loading fails, error shown to the user.
- If the wallet file is not encrypted (legacy/plaintext), the code will load it as it currently does

5.In-Memory Handling:
Once decrypted, sensitive wallet data like private keys will stay in memory until the app exits. We'll handle this carefully—zeroing out buffers (using crates like zeroize) as soon as they're no longer needed. Rust's ownership model helps here by keeping secret data tightly scoped.

Code Implementation:

- `src/wallet/encryption.rs`: Implements AES-GCM + PBKDF2 encryption and decryption.

```
use aes_gcm::{Aes256Gcm, KeyInit, Nonce}; // AES-GCM with 256-bit key
use aes_gcm::aead::{Aead, Payload};
use pbkdf2::pbkdf2_hmac;
use rand::{RngCore, rngs::OsRng};
use sha2::Sha256;
use zeroize::Zeroize;

const SALT_LEN: usize = 16;
```

```rust
const IV_LEN: usize = 12;
const KEY_LEN: usize = 32;
const HEADER_MAGIC: &[u8; 8] = b"CSWALLET";
const PBKDF2_ITERATIONS: u32 = 100_000;

/// Format: [MAGIC (8 bytes)] [salt (16)] [iv (12)] [ciphertext + tag]
pub fn encrypt_wallet(plaintext: &[u8], password: &str) -> Result<Vec<u8>, String> {
    let mut salt = [0u8; SALT_LEN];
    OsRng.fill_bytes(&mut salt);

    let key = derive_key(password, &salt);
    let cipher = Aes256Gcm::new(&key.into());

    let mut iv = [0u8; IV_LEN];
    OsRng.fill_bytes(&mut iv);

    let nonce = Nonce::from_slice(&iv);
    let encrypted = cipher
        .encrypt(nonce, Payload { msg: plaintext, aad: &salt })
        .map_err(|_| "Encryption failed")?;

    let mut output = Vec::new();
    output.extend_from_slice(HEADER_MAGIC);
    output.extend_from_slice(&salt);
    output.extend_from_slice(&iv);
    output.extend_from_slice(&encrypted);

    Ok(output)
}

pub fn decrypt_wallet(data: &[u8], password: &str) -> Result<Vec<u8>, String> {
    if data.len() < HEADER_MAGIC.len() + SALT_LEN + IV_LEN {
        return Err("Encrypted wallet format too short".into());
    }
    let (magic, rest) = data.split_at(8);
    if magic != HEADER_MAGIC {
        return Err("Wallet file is not encrypted".into());
    }
    let (salt, rest) = rest.split_at(SALT_LEN);
    let (iv, ciphertext) = rest.split_at(IV_LEN);
    let key = derive_key(password, salt);
    let cipher = Aes256Gcm::new(&key.into());
    let nonce = Nonce::from_slice(iv);
    cipher
        .decrypt(nonce, Payload { msg: ciphertext, aad: salt })
        .map_err(|_| "Decryption failed, wrong password or corrupted data".into())
}
pub fn is_encrypted_wallet(data: &[u8]) -> bool {
    data.starts_with(HEADER_MAGIC)
}
fn derive_key(password: &str, salt: &[u8]) -> [u8; KEY_LEN] {
```

```
    let mut key = [0u8; KEY_LEN];
    pbkdf2_hmac::<Sha256>(
        password.as_bytes(),
        salt,
        PBKDF2_ITERATIONS,
        &mut key,
    );
    key
}
/// Generate a random salt
pub fn generate_salt() -> [u8; SALT_LEN] {
    let mut salt = [0u8; SALT_LEN];
    OsRng.fill_bytes(&mut salt);
    salt
}
/// Generate a random IV for AES-GCM
pub fn generate_iv() -> [u8; IV_LEN] {
    let mut iv = [0u8; IV_LEN];
    OsRng.fill_bytes(&mut iv);
    iv
}
```

- `src/wallet/api.rs`: Wraps wallet save/load with encrypt/decrypt; detects encrypted headers.

```
pub fn save_wallet(
    path: &Path,
    wallet: &Wallet,
    password: Option<&str>,
) -> Result<(), WalletError> {
    // serialize → if Some(password) then derive_key+encrypt → write file
    ...
}
pub fn load_wallet(
    path: &Path,
    password: Option<&str>,
) -> Result<Wallet, WalletError> {
    // read header → if encrypted then derive_key+decrypt → deserialize
    ...
}
```

---

3. Local Encrypted Backup and Restore

Goal: Provide a safe way to back up the encrypted wallet file, preventing data loss if the main file is corrupted or deleted.

1.Automatic Backup on Shutdown:
For the maker, when `maker-cli stop` is called, `makerd` finalizes the wallet and copies the encrypted wallet to a backup path (e.g. `~/.coinswap/maker/wallets/backup/`)

2.Manual Backup via CLI:
We will add a CLI subcommand to both `maker-cli` and `taker` for manual backups. For example, `maker-cli backup` will instruct the running `makerd` to flush the current wallet state (if not already) and copy the wallet file to the backup location. Internally, this could be implemented as a new RPC call in the maker daemon (e.g. a `backup` command) . On the taker side, running `taker backup` will simply perform the backup operation (since the taker process can directly read its wallet file).

3.Performing the copy:

Performing the backup is simple—we'll copy the encrypted wallet file byte-for-byte to the backup path. Since it's already encrypted, no extra encryption is needed. We'll fsync the file to ensure it's fully written. Backups might be named wallet.bak or include a timestamp like wallet-backup-2025-07-01T15-30-00.dat.

4.Local Restore:

To restore from a local backup, we will provide a CLI command such as `maker-cli restore --file <path>` and `taker restore --file <path>`. This command will **not** automatically overwrite the current wallet without confirmation.The tool will load the backup file, attempt to decrypt it with the user's password (prompting if not provided), and if successful, replace the current wallet file with the decrypted data.

Code Implementation:

- `src/bin/makerd.rs`: Adds RPC handlers for `backup`/`restore`; triggers auto-backup on shutdown.

```
fn handle_shutdown() {
    // on SIGTERM: call backup_wallet()
    ...
}
rpc! {
    /// Copies encrypted wallet to ~/.coinswap/.../backup/
    fn backup_wallet() -> Result<(), RpcError> { ... }
    /// Restores from given file (decrypts with password, replaces current)
    fn restore_wallet(file: String, password: Option<String>) -> Result<(), RpcError> { ... }
}
```

---

4. Cloud Backup with AWS S3

Goal: Provide an off-site backup to mitigate local disk failures. Even if the user's entire machine fails, an S3-stored backup can be retrieved.

1.AWS S3 Bucket Setup:
We'll assume the user sets up an S3 bucket for cloud backups, with `s3_bucket_name` and `s3_region` set in the Coinswap config. Our Terraform script will ensure versioning is enabled. Backups will be stored under a fixed key like `maker-wallet.bak` (or with a user ID if needed). With versioning on, each upload creates a new version—no need to change the filename each time.

2.Credentials and Security:
We won't hardcode credentials. The AWS SDK will use standard methods—env vars, config files, or IAM roles. Users must have the right permissions (`s3:PutObject`, `s3:GetObject`, etc.). Everything goes over HTTPS, and since the file is encrypted before upload, data stays secure end-to-end.

3.Uploading Backup:
If cloud backup is enabled (`enable_cloud_backup=true`), the app will upload the encrypted backup after

local copy is saved. For the maker, `makerd` will handle this (either async or at shutdown). For the taker, it'll happen during the CLI command. We'll use AWS SDK's async API to stream the file and call `PutObject`.

4.Restore from S3:
We may add a `--from-s3` restore command later (`maker-cli restore --from-s3`, etc.). It would fetch the latest (or a specified version) using `GetObject`, save it temporarily, and run the usual restore logic.

Code Implementation:

- src/cloud.rs: is an helper module that handles uploading encrypted wallet backups to AWS S3 using the Rust AWS SDK.()

```
pub async fn upload_backup_s3(
    bucket: &str,
    key: &str,
    file: &Path,
) -> Result<(), S3Error> {
    // aws_sdk_s3::Client::put_object → await
    ...
}
pub async fn download_backup_s3(
    bucket: &str,
    key: &str,
    dest: &Path,
) -> Result<(), S3Error> {
    // aws_sdk_s3::Client::get_object → save to dest
    ...
}
```

---

5. Terraform for AWS S3 Provisioning (Versioned Buckets)

To aid users setting up the S3 infrastructure correctly, we will provide a Terraform configuration for creating the backup bucket with best practices. This ensures that even non-expert users can easily get a properly configured environment for cloud backups.It will be included that:

- Creates an S3 bucket with versioning enabled.
- Allows a user to run `terraform init && terraform apply -var="bucket_name=my-bucket"` to get a preconfigured environment.
- (Optionally) defines a minimal IAM policy that grants only `PutObject`/`GetObject` for a dedicated backup user.

Code Implementation:

- `infra/terraform/coinswap_backup_bucket.tf`: Creates versioned AWS S3 backup bucket.()

```
resource "aws_s3_bucket" "coinswap_backup" {
  bucket = var.bucket_name
  acl    = "private"
```

```
  versioning {
    enabled = true
  }
```

---

6. CLI and Configuration Additions

Configuration File Changes

We'll introduce or extend a `[wallet]` or `[backup]` section in `config.toml(both ~/.coinswap/maker/config.toml and ~/.coinswap/taker/config.toml)`:

```
[wallet]
encrypt_wallet = true

[backup]
backup_path = "/home/user/coinswap_backups/"
enable_cloud_backup = true
s3_bucket_name = "my-coinswap-bucket"
s3_region = "us-west-2"
```

CLI Extensions:

We'll extend both `maker-cli` and `taker` with new commands and flags using the existing `clap` setup.

Password Prompt: If `encrypt_wallet=true`, the user will be prompted for a password at startup. In `makerd`, this happens on the console (if interactive), or uses `--wallet-password` / env var if provided. In `taker`, the prompt will appear on every run unless overridden by a flag or env. We'll use something like the `rpassword` crate to hide input securely.
Commands:

- `maker-cli backup [--no-cloud]` – triggers local backup, optionally skipping S3.
- `maker-cli restore --file <path>` – restores from local backup; must confirm overwrite.
- Potentially `maker-cli restore --from-s3 [--version <id>]`.

Code Implementation:

- `src/bin/maker-cli.rs`: Adds `backup` and `restore` subcommands with user prompts. Interacts with `makerd` over RPC. ()
- `src/bin/taker.rs`: Adds internal handlers for `backup`/`restore`. Direct wallet file access.()

```
let cmd = clap::Command::new("backup")
    .about("local backup")
    .arg("--no-cloud");
let cmd2 = clap::Command::new("restore")
    .arg("<file>")
    .arg("--from-s3");
match matches.subcommand() {
    ("backup", sub)  => { /* call backup_wallet or skip S3 */ }
    ("restore", sub) => { /* call restore_wallet or fetch from S3 */ }
```

```
    _ => {}
}
```

7.Testing Plan:

1. Unit Tests:

We'll validate the core encryption logic by testing AES-256-GCM and PBKDF2 key derivation. These tests will ensure encrypted wallet data can be decrypted only with the correct password and that decryption fails with the wrong one. This guarantees our cryptographic implementation is secure and behaves as expected.

Demo Code Implementation:

- `src/wallet/encryption.rs` – tests for key derivation and AES-GCM encryption/decryption.()

```
#[cfg(test)]
mod crypto_tests {
    #[test]
    fn test_key_derivation() {
        // derive_key("pass", &salt) == expected_bytes
        ...
    }
    #[test]
    fn test_encrypt_decrypt_success() {
        // let (ct, iv, tag) = encrypt_wallet(pl, &key);
        // assert_eq!(decrypt_wallet(&ct, &key, &iv, &tag).unwrap(), pl);
        ...
    }
    #[test]
    fn test_decrypt_wrong_password() {
        // decrypt_wallet with wrong key → Err
        ...
    }
}
```

2. Integration Tests:

We'll simulate full backup and restore flows for both maker and taker. This includes creating an encrypted wallet, performing a backup, corrupting the original, and verifying that restoration brings the wallet back to its original state. Tests will also cover S3 upload triggers using mocks or a local emulator, ensuring cloud backup integration is sound.

Demo Code Implementation:

- `tests/backup_restore.rs` – simulates end-to-end wallet backup and restore for maker and taker.()

```
#[cfg(test)]
mod backup_tests {
    #[test]
```

```
    fn test_local_backup_restore() {
        // save encrypted wallet, corrupt orig, restore → matches original
        ...
    }
    #[tokio::test]
    async fn test_s3_backup_upload() {
        // mock S3, upload_backup_s3 → verify call
        ...
    }
}
```

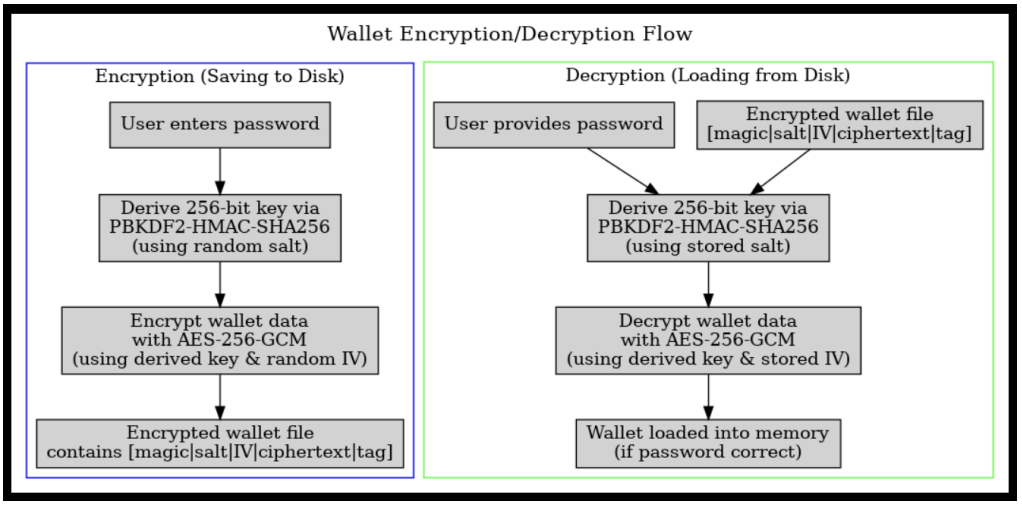
3. CLI Tests:

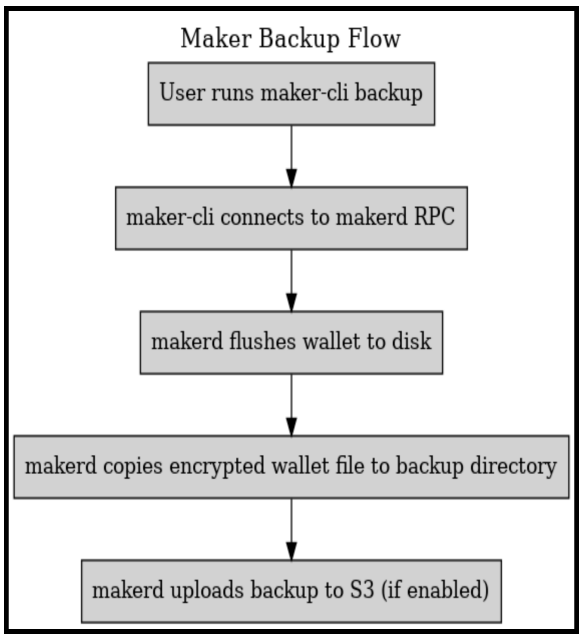
We'll test the new `backup` and `restore` subcommands using simulated CLI inputs to ensure correct parsing and argument validation. These tests confirm that the commands behave reliably, provide clear error messages, and integrate cleanly into the existing maker-cli and taker interfaces.

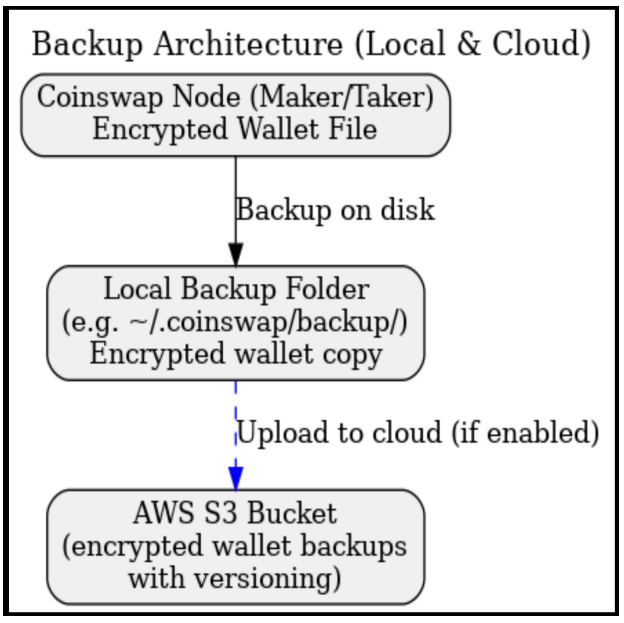
Demo Code Implementation:

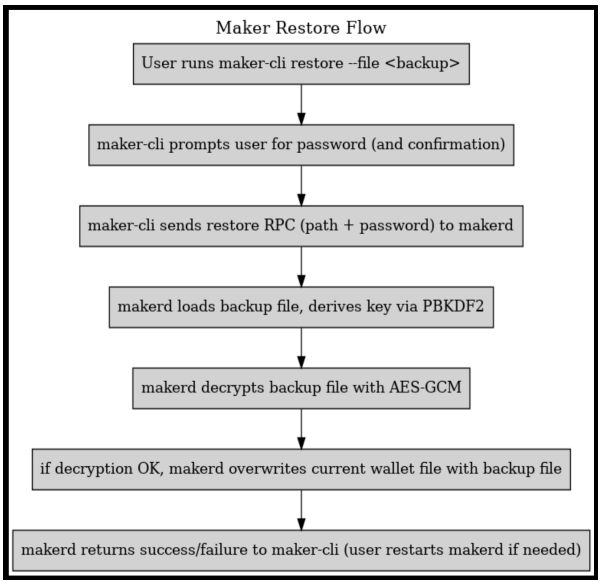
- `tests/cli_backup_tests.rs` – tests `maker-cli` and `taker` backup/restore commands via subprocess calls.()

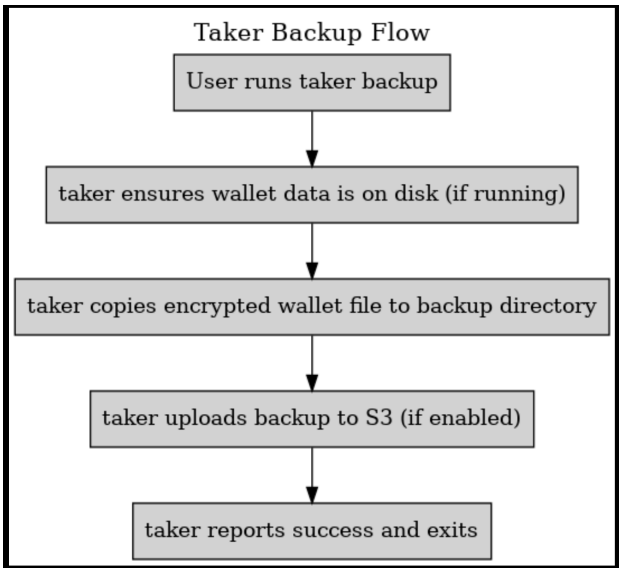
8.Flow Diagrams:

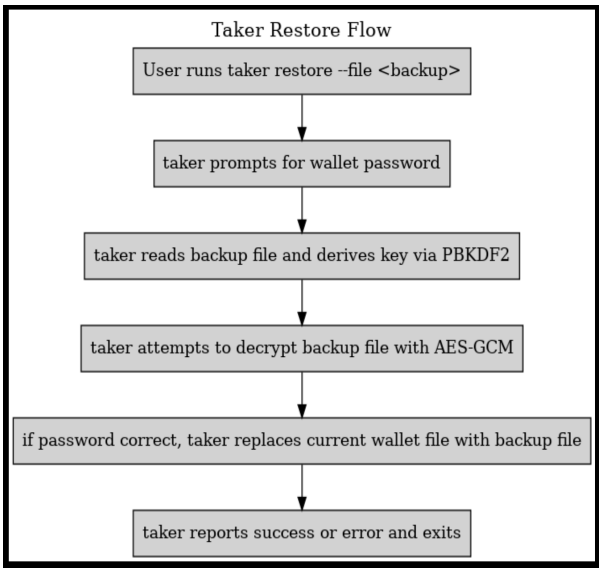












Project Timeline:

Weeks 0–1 (Community Bonding):
I'll finalize the technical design with my mentor, explore the relevant Coinswap modules, and set up local and AWS environments needed for development and testing.

Weeks 1–2 (Wallet Encryption):
Implement AES-256-GCM encryption with PBKDF2 password derivation. This includes prompting the user for a password and modifying wallet load/save logic to support encrypted files.

Weeks 3–4 (Local Backup & Restore):
Add logic to back up the encrypted wallet locally, triggered on shutdown or via CLI (`backup`/`restore` commands). Implement password-based restore with proper safety checks.

Weeks 5–6 (Cloud Backup with S3):
Integrate AWS S3 uploads using the Rust SDK and provide a Terraform script to set up a versioned S3 bucket. Ensure cloud backup works after local backup completes.

Week 6 (Midterm Evaluation):
By midterm, encryption, local backup/restore, and basic cloud integration will be complete and demo-ready.

Weeks 7–8 (CLI Enhancements):
Polish CLI UX: add flags like `--wallet-password`, confirmation prompts, and improve error handling. Optionally support restore directly from S3.

Weeks 9–10 (Testing & Bug Fixes):
Write unit and integration tests for encryption, backup, and restore flows. Fix edge cases like corrupted files or incorrect passwords.

Week 11 (Polish & Docs):
Finalize code quality, logs, memory zeroing, and write documentation with examples. Prepare demo materials.

Week 12 (Final Submission):
Submit the full implementation, final report, and assist with review or merge process.

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 makes Coinswap more secure and reliable by encrypting wallet files and adding automated backups. It helps users safely run nodes without risking key loss or compromise. These changes make the protocol more usable, especially for long-term makers and real-world adoption.

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.

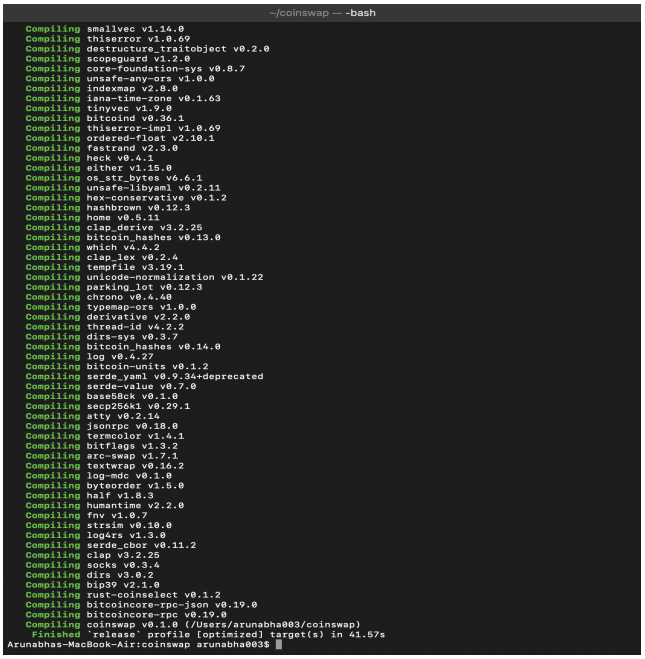
When I looked into past SOB organizations, Coinswap immediately caught my eye. After running the demo, I was hooked—I dove into the codebase and decided to contribute. I believe I’m an ideal candidate because I’m genuinely committed to getting Coinswap production-ready. I’ve worked with Move (a Rust-like language) before.Still I consider myself as a rust beginner, and on a continuous process of learning more. 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 am currently working on another one.
1.https://github.com/citadel-tech/coinswap/pull/455 (merged)
2.https://github.com/citadel-tech/coinswap/pull/452 (merged)

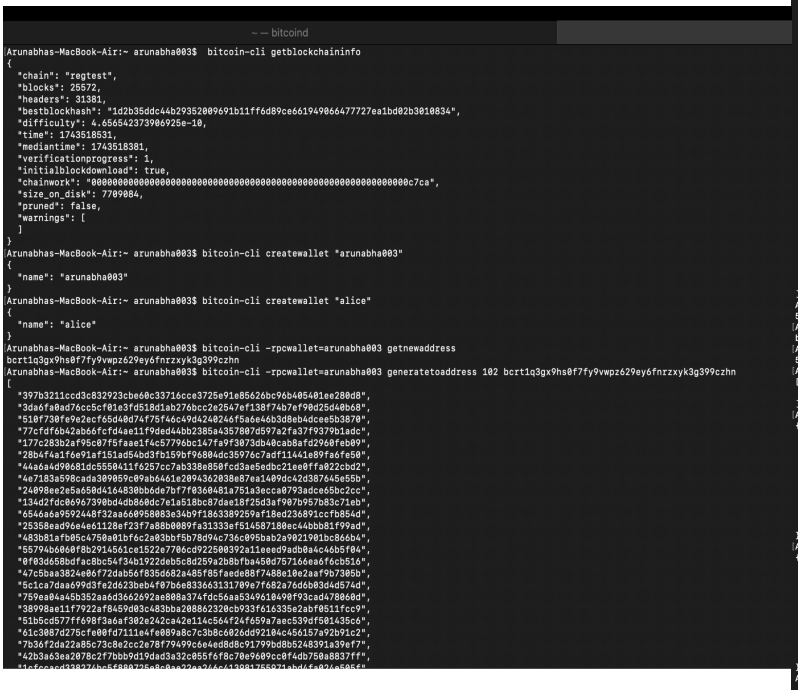
3. https://github.com/citadel-tech/coinswap/pull/486 (open PR)

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.



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": "0000000000000000000000000000000000000000000000000229b52f27cf44393ebc",
  "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$
```