Super Private P2P onchain pipeline for Robosats

Personal Information

- **Name**: Aarav Mehta
- **Email**: aarav.mehta.mat22@itbhu.ac.in (college mail id) / aarav05mehta@gmail.com (personal mail id)
- **GitHub**: https://github.com/aaravm
- **Phone no. :** +91 9653321136
- **Major:** Mathematics and computing
- **University:** Indian Institute of Technology, Varanasi
- **Time zone**: UTC+5:30

Motivation

I am a 2nd year undergraduate, studying at IIT (BHU) Varanasi. I have been doing some kind of software development since high school (mainly web development). But after coming to my college, I was introduced to a whole new world and I have learnt about so many new and interesting languages, frameworks and fields.

I was introduced to the world of open source during December 2023, and I fell in love with the idea of open source- to prevent some private company owning an important library, instead it being maintained in a community, where anyone can come and check the work of the community and contribute to it.

I started my journey in Web3 when I joined the programming club of my college. I learnt about the basics, and found it really interesting. I decided to explore this field further then. I did a course on solidity, and built a few websites on ethereum. Since then, I have looked at a lot of different protocols and other blockchains like the lens protocol, Aptos blockchain, Dojima Network and so on.

I learnt about Summer Of Bitcoin from the club, as one of my seniors had previously qualified it. I was instantly interested in learning more about it. My journey with Summer of Bitcoin has been amazing so far, with me having learnt a lot, firstly from the bootcamp, where I completed the book Grokking Bitcoin, and then the assignment, where I was successfully able to verify and mine raw transactions.

I learnt about Robosats from the organizations list from Summer of Bitcoin site and instantly felt a liking towards it as it is responsible for allowing p2p trades to become much easier, and allowing more mass market adoption of cryptocurrency.

I chose this particular project, because of my past experience with a similar project in my programming club of the college, where I learnt a lot about data pipelining. I also felt that the community in Robosats is very helpful, and eager in welcoming new contributors, like when they gave me a simple PR, https://github.com/RoboSats/robosats/pull/1232 which allowed me to get introduced to Robosats.

During this period, I currently have no other commitments. I'll have summer vacations which I will be spending at my hometown. I intend to remain connected with the community and contribute to it even after SoB and expand this project to a next level.

Technical Skills

I am currently a sophomore at IIT (BHU) Varanasi in the field of maths and computing, pursuing my B.tech+ M. tech degree, with a CGPA of 9.68 out of 10. I have completed the courses of Discrete Mathematics, Computer System Organisation, Computer Programming, Data structures and algorithms, Probability and Statistics, Linear Algebra with an A grade.

I am always exploring and learning different things and participating in hackathons, competitions etc. I have participated in a lot of web3 hackathons (I have added the links of some of my hackathon projects below). One of my best achievements has been that I got a silver medal in the prestigious month long hackathon, Inter IIT Tech Meet 12.0 (held by IIT Madras) in the field of web3, where I was responsible for the backend, written in move, a rust based language.

I am also an open source contributor, with 2 merged pull requests in the optuna library, written in python: https://github.com/optuna/optuna/pull/5322 and https://github.com/optuna/optuna/pull/5306

Some of my past projects have been: (For more details, click the link)

- Futures on chain
- Social Chain 3.0
- HealthCare 3.0
- Zip(Web Assembly)

Contents

Project Details

Brief Background

Robosats

RoboSats is a simple FOSS tool to privately exchange bitcoin for national currencies. Robosats simplifies the peer-to-peer user experience and uses lightning hold invoices to minimize custody and trust requirements. Anyone can be a robosats user and any lightning node can become a robosats provider, no barriers no costs.

What is the Taproot upgrade?

Bitcoin Taproot update is the most important upgrade the cryptocurrency has experienced since 2017, when Segregated Witness (SegWit) was activated. Like SegWit, the Taproot upgrade aims to improve the privacy and efficiency of the network but on a larger scale and with a potentially more significant impact expected over the years.

Bitcoin's standard process before the Taproot implies that transactions get verified individually by validating a digital signature — which is like a user's fingerprints — against a public key. The Taproot upgrade lets multiple and complex signatures like multi-signature wallets be aggregated and verified together, rather than individually.

The main change that will enable this significant transition is the implementation of Schnorr signatures over the Elliptic Curve Digital Signature Algorithm (ECDSA). The ECDSA algorithm creates a signature from the private key that controls a Bitcoin wallet and verifies that the rightful owner carries out the transaction.

The Taproot also enhances privacy since there won't be more distinction between multi-signature and single-signature transactions. This way, it becomes more difficult to identify each participant's transaction inputs where the private data is stored.

Taproot introduces Taptrees, a feature that reduces the size of transaction data and ensures only necessary information is revealed on the blockchain, thereby preserving privacy. With Taproot, multisig transactions are also more private as unused spending conditions are hidden.

Motivation for the Project

There are some kinds of trades that are not possible due to the short expiry times of routable HTLCs. In addition, the trust properties of p2p lightning are somewhat weak. We can make use of taproot contracts to build a super-private p2p onchain pipeline, that solves these problems.

Project Goal & Milestones

The aim of this project is to make use of taproot contracts to build a super-private p2p onchain pipeline that will look just like a basic P2TR if the trade goes well (no disputes). We will use BDK to build the transaction, and then build our functionality into a Rust wasm library that will run on the client's browser for maximum sovereignty.

The following milestones can be defined:

- **Milestone 1 (MS1):** Create a basic descriptor (using only 2 paths) and try to mint a p2tr transaction
- **Milestone 2 (MS2):** Create the functionality of creating a wallet, and adding functionalities to the wallet like viewing balance, transferring funds from and to the wallet
- **Milestone 3 (MS3):** Create the actual descriptor with all 4 paths and check if the transaction is being published
- **Milestone 4 (MS4):** Create the CLI that allows users(client) to interact with the repository, and create p2tr transactions
- **Milestone 5 (MS5):** Compile the repository into wasm and bundle it as a npm package
- **Milestone 6 (MS6):** Understand the current Robosats client and coordinate infrastructure, to understand how to fit the npm package into it
- **Milestone 7 (MS7):** Build the new package into the current robosats client and coordinator infrastructure
- **Milestone 8 (MS8):** Produce comprehensive documentation and engage with the Robosats community

Proposed Model

Transaction contract can be defined using Taproot Descriptors, which can then be built upon by adding features like creating a Bitcoin wallet, generating addresses, receiving funds into the wallet, constructing a transaction, signing the transaction, combining multiple signed transactions, finalizing the transaction, and broadcasting it to the blockchain.

Transaction Contract:

Case	TR/MAST	Signers	Sats move to	Onchain trace
1. Cooperative Success	Default spending path	Buyer and seller	escrow to buyer, fee to coordinator	P2TR: 1 input 2 outputs
2. Cooperative Cancellation	Script_A	Buyer and seller	escrow back to seller	Reveals TapLeaf
3. Seller wins dispute	Script_B	Seller and coordinator	escrow back to seller	Reveals TapLeaf
4. Buyer wins dispute	Script_C	Buyer and coordinator	escrow to buyer, fee to coordinator	Reveals TapLeaf

The easiest and safest method to define the p2tr contracts is descriptors (https://bitcoinops.org/en/preparing-for-taproot/#taproot-descriptors). Taproot descriptors provide a standardized way for wallets to store information needed to create, scan, and spend from addresses, enhancing interoperability and convenience. Descriptors have been updated to work with taproot, allowing for keypath and scriptpath spends. For example, the `tr(<key>)` descriptor can be used for single-sig, and `{`

`{pk(<b_key>),pk(<c_key>)} , {pk(<d_key>),pk(<e_key>)} )` for scriptpath spending. h
Here, this syntax allows specifying the contents of a binary tree. For example, `{ {B,C} , {D,E} }` specifies the following tree:

```
Internal key
    / \
   /   \
  / \ / \
  B C D E
```

To generate the transaction contract, let us consider a simple case. Let there be 2 people, Alice and Bob. Let the 2 different policies be: serieshd.cc
Either (1) Alice takes the money after waiting 1 day, or (2) Bob takes the money whenever he wishes to, by revealing preimage_hash?
The descriptor can be made like this:

```
// Alice's policy: Spend after 1 day
let alice_policy_str = format!("older(144)");  // 144 blocks = 1 day
(assuming 10 min block time)
let alice_policy =
Concrete::<String>::from_str(&alice_policy_str)?.compile()?;
let alice_tap_leaf = TapTree::Leaf(Arc::new(alice_policy));

// Bob's policy: Spend at any time with preimage_hash
let bob_policy_str = format!("hash160({})", preimage_hash);
let bob_policy =
Concrete::<String>::from_str(&bob_policy_str)?.compile()?;
let bob_tap_leaf = TapTree::Leaf(Arc::new(bob_policy));

// Combine the two policies into a Taproot tree
let tap_tree = TapTree::Tree(Arc::new(alice_tap_leaf),
Arc::new(bob_tap_leaf));

// Construct the Taproot descriptor
let descriptor = Descriptor::new_tr(dummy_internal, Some(tap_tree))?;
println!("{}", descriptor);
```

Reference:
https://github.com/danielabrozzoni/multisigs_and_carrots/blob/master/src/generate_descriptor.rs

Client CLI

Step 1: Generating private keys:

```
    // Step 1: generate a WIF
    let private_key: GeneratedKey<_, miniscript::Tap> =
PrivateKey::generate_default()?;
    let mut private_key = private_key.into_key();
    private_key.network = Network::Testnet;

    // Step 2: print the xpub to stdout
    let secp = Secp256k1::new();
    let public_key = PublicKey::from_private_key(&secp,
&private_key);
    println!("Public key: {}", public_key);

    // Step 3: save the WIF in a file
    let mut file = File::create("key.txt")?;
    file.write_all(private_key.to_string().as_bytes())?;
```

Alternative #2: Directly, internally pass the public key to the coordinator, and display the private key generated to the user

Coordinator CLI

Step 1: Create a wallet, that allows the coordinator to hold the funds deposited by the client

```
let wallet = Wallet::new(
        // TODO: insert your descriptor here
        "tr(descriptor_made)",
        None,
        Network::Testnet,
        MemoryDatabase::new()
```

```
)?;
```

Step 2: Choosing the policy whose condition has been met: [TxBuilder in bdk::wallet::tx_builder - Rust] (using BDK)
For example: using the second policy here to send the funds in the coordinators' wallet to Bob.

```
let mut path = BTreeMap::new();
path.insert(wallet_policy.id, vec![1]);

let mut tx_builder = wallet.build_tx();
tx_builder
    .drain_wallet()
    .drain_to(BOBs_ADDRESS.script_pubkey())
    .fee_rate(FeeRate::from_sat_per_vb(3.0))
    .policy_path(path, KeychainKind::External);

let (psbt, _details) = tx_builder.finish()?;
```

Step 3: Publish to the p2tr transaction to the blockchain:

```
let secp = Secp256k1::new();
let psbt = base_psbt.finalize(&secp).unwrap();
let finalized_tx = psbt.extract_tx();
dbg!(finalized_tx.txid());

let blockchain =
EsploraBlockchain::new("https://blockstream.info/testnet/api", 20);
dbg!(blockchain.broadcast(&finalized_tx));
```

NOTE: This can be expanded upon, and more conditions can be added upon this on the basis of the exact details of the transaction, the code provided above is just a basic example of how the code will look like.

UI/ UX of the Command Line Interface tool:

The Command Line interface can be made by using the clap library to parse the input and using match to call the corresponding function:

```
use clap::{App, Arg};
fn main() {
   match matches.subcommand_name() {
        Some("create-key") => {
       }
        Some("send-funds") => {
       }
     // Other functionalities we want to use
    }
}
```

Compiling

Compiling the code can be done using wasm-pack, by first installing wasm-pack and then running the following command to generate the wasm package to the pkg directory:

```
wasm-pack build --target web
```

This command will compile the Rust code to WebAssembly and generate a `pkg` directory containing the Wasm binary and JavaScript bindings.

To export it as a npm module, I will create
index.js:

```
const wasm = require("./wasm_package");
module.exports = wasm;
```

And export it by defining an appropriate package.json

Timeline

I propose the following timeline for completing the milestones of this project. I believe I will require less time than I have outlined, and therefore I hope to work on the stretch goal of confidential computing and hopefully complete it as well

- **Program Kick-Off and Onboarding:** May 8 - May 15

In this period I will be in contact with my mentors to set up communication channels and work policies. Furthermore I will be focusing on getting the project basics set up. This includes:

  - Working with my mentor, and discussing with the community to any proposed change in the project proposal
  - Setting up the repository on github
  - Setting up the workspace environment for the project locally

Phase 1

- **Week 1:** May 16 - May 22
    - Create a basic descriptor (using only 2 paths) and try to mint a p2tr transaction(**MS1**)
    - Add documentation on how to make a basic p2tr transaction
- **Week 2:** May 23 - May 29
    - Create the functionality of creating a wallet, and adding functionalities to the wallet like viewing balance, transferring funds from and to the wallet(**MS2**)
    - Add tests to make sure the correct balance is returning, and the funds are transferred
- **Week 3-4:** May 30 - Jun 12
    - Create the actual descriptor with all 4 paths and check if the transaction is being published(**MS3**)
    - Complete the integration and make the basic version of the library, which can mine the transactions

- **Week 5-6:** Jun 13 - Jun 26
    - Create the CLI that allows users(client) to interact with the repository, and create p2tr transactions(**MS4**)
    - Add tests to make sure the right output is being returned for the users and add documentation on how to use the CLI
- **Week 7:** Jun 27 - Jul 3
    - Complete any remaining tasks for mid-term evaluation
- **(Phase 1 evaluation):** July 1 - July 5

  Deliverable: A demonstrator CLI tool (client-coordinator-client) that is capable of doing p2p trades using TR/mast features by connecting to another client and a coordinator.

Phase 2

- **Week 8:** Jul 4 - Jul 10
    - Compile the repository into wasm and bundle it as a npm package(**MS5**)
    - Add tests to check if the wasm works as intended
- **Week 9-11:** Jul 11 - Jul 31
    - Understand the current Robosats client and coordinate infrastructure, to understand how to fit the npm package into it(**MS6**)
    - Allow the npm package to run on the client's browser for maximum sovereignty.(**MS7**)
- **Week 12:** Aug 1 - Aug 7
    - To wrap up any unfinished tasks or deal with unexpected delays, I intend to reserve this week as a buffer.
    - Ask the community for any suggestions on how to improve the project, and discuss any potential changes or additional features to be added(**MS8**)
    - Do the finishing touches in the project

- **Week 13:** Aug 8 - Aug 15
    - Preparing a short report on the project
    - Preparing a short slide deck with intro/background/motivation, results/implementation, discussion/outlook (3-5 slides)
    - Posting about it on LinkedIn and on Medium
- **Final submission: Aug 26**

  Deliverable: The final submission will be a super-private p2p onchain pipeline that will look just like a basic P2TR if the trade goes well (no disputes). It will use BDK to build the transaction, and then build the functionality into a Rust wasm library that will run on the client's browser for maximum sovereignty.

Availability

I will be available throughout SoB and I have no other commitments during this time. I will be devoting about 40 hours a week to my goals during SoB with Robosats. I will always be available through email and telegram. If I am behind my proposed timeline, I will work overtime for the next few days to make things go according to the plan.

Post SoB

Being part of such a collaborative and welcoming organization is a great opportunity for me to learn more and contribute. So, I have already planned to contribute more to the organization even after the SoB period, and also work on the stretch goal if it is not completed. I'll be an active member of the community and keep contributing to the organization

References

- GitHub - danielabrozzoni/multisigs_and_carrots: 🥕 Create a multisig with taproot and spend from it using BDK 🥕
- https://bitcoindevkit.org/blog/2021/11/first-bdk-taproot-tx-look-at-the-code-part-1/
- https://github.com/bitcoin-core/btcdeb/blob/master/doc/tapscript-example-with-tap.md
- https://dev.to/eunovo/a-guide-to-creating-taproot-scripts-with-bitcoinjs-lib-4oph
- https://bitcoinops.org/en/preparing-for-taproot/#taproot-descriptors
- https://www.youtube.com/@bitdevsla268/featured