Motivation and overview

Because most chains share a codebase and history, the cryptocurrency ecosystem is inextricably linked. Despite this, communities and tools remain fragmented. Most user-facing tools are forked from other ecosystems and poorly maintained. As a result, vulnerabilities follow tools across ecosystems. For instance, Electrum, a widely used wallet, ported to over a dozen chains, was found to have a vulnerability that required all fork maintainers to release an upgrade, and all users to install it.

We built Riemann to unify tooling across chains. Riemann is a library for transaction constructions on Bitcoin-based blockchains. It accepts human-readable inputs and serializes transactions from them. Riemann turns single-asset tools into natively multi-asset tools. It can be used to build shared infrastructure across ecosystems. Currently, we support with over 20 live networks as well as dozens of testnets.

Riemann leverages the shared heritage of Bitcoin-based chains. The majority of serialization logic is handled by a shared core. Supporting transparent transactions is trivial, as Zcash version 1 transactions are nearly identical to Bitcoin transactions. However, JoinSplits, introduced in Sprout's version 2 transactions, are a large departure. In addition, the Overwinter and Sapling serialization formats will continue to diverge from standard Bitcoin transactions. Transaction expiry, shielded inputs and outputs, and other format changes will prevent us from using the shared core. New data structures need to be created to reflect the new Overwinter serialization formats as well as the upcoming sighash changes.

We intend to upgrade Riemann to support transaction versions 2-4. After the upgrade, users should be able to easily parse and inspect any transaction from any official Zcash version, and quickly construct valid transactions from arbitrary inputs, provided they have the necessary signatures and proofs.

Technical approach

Riemann treats transactions as objects thinly wrapping an underlying bytearray. We will implement new objects for transaction versions 2, 3, and 4. These objects will provide convenient ways to inspect transactions, as well as abstractions for important properties like sighash types. An example of the approach can be found in the Decred-specific data structures here.

Once completed, we will integrate them into the existing transaction builder interface, as well as other simplified interfaces. Wherever possible, developers will be presented with a unified interface that abstracts away Zcash-specific concerns by setting rational, overridable defaults. For example, expiry height will be set to 0 by default. For most use cases, this means code can be directly reused between Bitcoin, Zcash, and other coins.

We will also create Python bindings for the zcash-sprout-verifier library.

Team background and qualifications

Team:

Summa

James Prestwich, founder @ Summa former COO @ Storj

Rachel Rybarcyzk, blockchain engineer @ Summa

Qualification:

James has worked full time on cryptocurrency for four years. He is a founder of Summa and Storj, and has contributed code and time to Chia, Least Authority, and several other projects.

Rachel is a blockchain engineer at Summa. Previously a Python developer at Northrop Grumman, she is completing a masters degree in aerospace engineering. Rachel operates a small mining rig in her apartment.

Evaluation plan

We are successful when we:

- Can reliably create consensus-correct transactions of version 2, 3, and 4
- Can import and inspect any transaction from any Zcash main or testnet
- Create Python bindings for zcash-sprout-verifier to validate Sprout and Overwinter JoinSplits
- Create a separate tool for verifying proofs
- Have test coverage on the above

To help the foundation evaluate our results we will:

- Deliver sample scripts to import and inspect transactions from live Zcash networks
- Deliver sample scripts to validate JoinSplits from live Zcash networks
- Deliver sample scripts to create transactions using Riemann
- Create on-chain transactions using Riemann

Security considerations

Riemann is a developer tool, not an application. It is released under the LGPL public license. It does not communicate with any other library or tool. It does not handle private keys or other sensitive data. It is almost stateless, and data structures are immutable. Everything Riemann

processes is intended to be broadcast on the network. It is difficult to see privacy or security implications in our work.

Schedule

- We will support Overwinter before its launch
    - We are currently on track for the June 25th launch
    - A work-in-progress pull request can be found here
- We expect to support Sprout proof verification before Overwinter's launch
- Timing of support for Sapling is contingent on the finalization of its spec
    - We expect to be able to support it before launch
- If a rustlang verifier for Sapling proofs is available, we will endeavor to support it before Sapling's launch

Budget and justification

We expect to spend approximately 100 hours up-front development time. Our primary cost is lost revenue from declining other projects. Our average rates for contracting are $250/hour. We are willing to work below cost as well as provide long-term maintenance and upgrades. We are looking for a project budget of $20,000.

Email address(es) for direct contact

james@summa.one