This post summarizes short-term goals for Circles Entropy. We provide some background about CiclesUBI in its current form for enough context to be able to reflect on the general problem statement as well as problems with the current implementation. We then outline a short-term approach to the problem by outlining an architecture centred around solvers that run trusted execution environments (TEEs) for solving.

CirclesUBI

We start by stating some points that are not entirely clear in the current CirclesUBI [witepaper] (Whitepaper | Circles UBI | Handbook).

CirclesUBI acts as a payment system that relies on the web of trust where every user has their own personal Circles(CRC) token minted at the rate of 1 CRC per hour. Trust has clear definition in the context of CirclesUBI and is explicitly stated by users. Creating a trust connection between A and B (B trusts A) means that:

- B now accepts A-tokens as a part of a payment
- B now accepts that during a payment done by any user in the system A-tokens (together with any number of other token types B accepts) could be swapped automatically with any token B has (including B-tokens) and sent to any users that accept corresponding tokens, as long the sum of incoming tokens equals the sum of outgoing tokens. This is known as transitive transfer.

Trust limits

One implementation detail of CirclesUBI is the “trust limit”, a user-configurable threshold (set to 50% by default) that limits maximum amount of other users tokens present in an account at any time. It was implemented as a quick safeguard for users but has been proven to be problematic when dealing with large transactions. In future iterations of CirclesUBI, fully programmable trust limits to allow users to express their preferences would make more sense.

Max-flow and capacity graph

In order to maximize spendable Circles between users, the Max-flow algorithm is used to calculate maximum payment capacity between two users. Over time as CirclesUBI users and their trustees make transactions and receive payments we arrive at a situation where every user holds an array of different tokens from users they trust. This further increases dependency on highly-interconnected trust connections to maximize tokens a user is able to spend in the network. Thus Max-flow is not executed on a web of trust but rather on a structure we call “capacity graph” that is derived from WoT graph + state of all user assets allocation.

The payment capacity of an edge is the number of tokens possessed by the source node that are accepted by the target node. A node’s token never leaves the star centred at that node. Since a token is free to move around in its star, it can be freely swapped around to increase capacity for a certain path in the graph. In practice, there is an upper limit to how many tokens can be swapped in such a manner.

Abstract problem statement

Compute on a graph, minimizing for

1. trust assumptions between solvers and users
2. single points of failure
3. statistical disclosure to observers (including solvers)
4. disclosure of the graph structure itself
5. Computational resources

Concrete problem statement

Compute maxflow over distributedly-held graph data. The graph in this case is a directed graph where nodes are users and an edge between two nodes corresponds to a possible flow path for tokens. Flow of an edge is the number of tokens in possession of the source that are trusted by the target.

Addressing the problem

In the current implementation, transaction data is published on-chain as a series of pairwise token transfers. Since there is no account hiding, it reveals data about the complete trust path of a transaction.

The key challenge is data being discerned by solvers.

- Solver/user set intersection: We can't prohibit solvers from also being users that make payment requests, which means if there is no rate liming of requests, there is high potential for statistical disclosure attacks from solvers.
- Graph data leakage: Path finding algorithms should not leak information about the graph. This not only implies hiding graph data itself but hiding execution. Memory access should be data-independent or oblivious.

An MPC-based approach where parts of the data are held by several agents and invoked when necessary opens up other research questions about when-to-invoke-whom and fine-grained trust assumptions (e.g. a solver who has access to a user's trust data has different privileges than a solver who doesn't).

FHE does not solve the problem of statistical disclosure attacks by solvers. Furthermore, current implementation of FHE are too slow for practical use.

Current approach

We describe our current approach, that leverages TEEs to hide computation from the solver. A sketch of the high-level architecture can be found here.

A solver runs a CometBFT node, which has the ability to offload (parts of) the application logic to a secure enclave (the TEE). In particular, the path-finding happens on the enclave. Once a path with sufficient flow-capacity with respect to the transfer request is found, the request is considered solved. The transaction is then settle on some asset chain (e.g. Namada).

The enclave has access to an encrypted database that it may query for graph data during the computation. It generates a zero-knowledge proof of having found a path with maximal flow subject to restrictions described in the first section. Flow capacities are calculated based on assets that have been bridged to the app-chain from the asset chain.

This gives a protocol that needs consensus twice (once while solving and once while settling). It may be argued that it would be better to have consensus once rather than twice, but separating solving from settlement provides modularity. Using a settlement layer like Namada means users can benefit from the multi-asset shielded pool.

Current status

We are looking at various TEE solutions (SGX/TrustZone) and efforts are ongoing to setup some testing infrastructure that could be used to incrementally prototype zk-proof generation on a TEE.

Parallely, we are looking at approaches to max-flow that may be best suited for writing efficient circuits. The current implementation uses a variant of the Ford-Fulkerson algorithm, but a linear constraint-solving approach, apart from being more general, may have merits in the circuit setting. While proving that a computation has taken place subject to constraints requires a proof of every step of the computation, we are currently hoping to do atleast some of the computation outside of a circuit for efficiency gains.

We will do another post on a more long-term approach to solving this problem, which involves Taiga integration, an arguably better network topology and most importantly, no implicit trust assumptions involving large chip-manufactuting corporations.

In the meantime, questions or further discussion of any topic touched upon hitherto is appreciated.