[

1156×718 79.9 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/074ef811114629f939b0c21a8048c1888827b36d.png)

All content herein is sourced from this awesomevideo by Yulia and follow-up conversations with her on the topic.

- Users of Taiga can build applications on top of Taiga
- Taiga makes sure they work as they should
- Taiga is shielded provides and both data and function privacy

Taiga uses a notion of VPs which are declarative smart contracts. In more well known smart contract architectures, you write down what you want to do and execute. With VPs you describe the final State or the properties it should have and Taiga makes sure it's satisfied and doesn't care how it's done. For example, when you go to a restaurant, you don't necessarily (though you may) care about how food was made. Instead, you order off the menu and let the kitchen staff worry about the making the meal. Conversely, when you prepare food at home, you have to take an interest in how the food is made.

Taiga Overview

- Taiga provides atomic state transitions of arbitrary complexity
- Intents are user preferences, users describe what they want and Taiga tries to make sure their intents are satisfied with the help of a c/d layer, so matching intents is taken care of
- Application contains 2 main parts, application state and application logic
- Application state is built from all the resources (formerly notes) that the application owns
- Application logic is expressed as a validity predicate
- VP Of the application defines the application
- Atomic state transitions (of arbitrary complexity)
- Many systems when they want to provide transactions of arbitrary complexity can't really achieve it with one transaction
- They require a sequence of transactions to achieve a certain state
- This is not good because the system can end up in an intermediate state if something goes bad
- Taiga makes sure there is one transaction that takes state to the desired one and if something goes bad it stays in the initial state

The recipe for atomic state transitions

What is a valid Taiga Tx

? It has to satisfy 2 properties:

1. VPs of all involved apps are satisfied. What makes an application to be involved? If a transaction tries to change the resource

that belongs to an application, an app needs to approve it, all changes are authorized by VP of apps.

1. A Tx

needs to be balanced. All resources

carry some numeric value. A balance transaction is a Tx

for each application type, the total value of the input resources

= the total value of output resources

.

- We want to combine partial transactions together until their total value is balanced

- Valid transaction = validity predicates are satisfied and its balanced, such transactions can be published on a blockchain
- The intent application is just an application that works in a certain way
- The user who wants the intent to be satisfied sends the intent to the intent gossip network, and they add an application resource

to their intent

- While this resource

exists, the transaction cannot be balanced. The amount of input resources

of these applications does not match the amount of output resources

. We create this resource

, but don’t destroy it.

- Once all the users in this Tx

destroy their resource

, the transaction is balanced and can be published on the blockchain

- The application makes sure the transaction is not balanced until the resource constraints are satisfied

Solvers

- Solvers are the actors that make everything possible
- They build transactions, receive intents, and match them together
- Solvers when they receive PartialTxs

and PartialTx

sets, they combine them together and update the set of PartialTxs

until it’s all balanced, and once balanced can publish it

- A PartialTx

cannot be published on blockchain

- Intent application makes sure transactions are partial as long as transaction is not satisfied
- Solvers try to update their transactions until satisfied
- Allows to create atomic state transitions of arbitrary complexity

3 Party Barter Example

[

1840×860 321 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/355d587b8d51044f07c04020b0f6f41f9b1e61e1.jpeg)

The above is a three party barter example. Three users who want to get something, and they have something, but they can satisfy one of the others but need to be combined in a cycle. Every user - Alice, Bob and Charlie create their first PartialTx

, they also create a resource

.

- These partial transactions go to the first solver
- First solver receives 2 and naturally decides to give Bob’s dolphin to Alice. Because Alice’s intent is satisfied, they can spend Alice’s resource

. The total balance of this partial transaction set is not 0, so it needs to be solved further

- PartialTx

goes to next solver, who also receives Charlies PartialTx

. Sees that Charlie wants a star and Alice had a star, so a solver sends star to Charlie. The solver sends the tree to Bob.

- Now because everyone is satisfied here all resources

are spent or destroyed, everyone is satisfied and the Tx

can be spent on the blockchain.

Privacy

Privacy

What

How

Data

Resources

Verifiable encryption, Resource commitment

Transaction

(resources, participants, quantities, etc.)

ZKP, hashes, blinding

Function

Validity predicates

Recursion

We achieve both data and function privacy. Resource content is the value of the application type. We keep resources

encrypted. We use verifiable encryption to make the resources

decrypt-able. To establish the existence of the resources

, we use resource commitments. We publish resource commitments and people know some resource

just started to exist. For transaction data what resources

are being transferred, created or destroyed, who participates, value of resources

and applications for that we use ZKPs to prove that the state transition was correct without revealing the state transition, and also hashes and blinding to keep everything private.

For function privacy, which refers to VPs because VPs represent application function. We use ZKP recursion for that, the proof of a VP is hidden inside another proof, so the outer verifier can't really know what's inside the VP.

The proving system is Halo 2 built by ZEC we considered Plonk based and many polynomial commitment schemes, but Halo 2 supports recursion and accumulation, has some helpful gadgets; e.g. elliptic curve operations & Hashes. Halo 2 does not require a trusted setup, which when you have a lot of circuits it's helpful.

- The initial PartialTx

is generated locally, next partial transactions are generated by the solver

- In principle, anything that provides proper at least two-layer recursion can be used instead of Halo2.
- Blake2 is used to hash for field-independent commitments
- Poseidon is used for many things
- Sinsemilla commitment scheme for resource commitments (same as Orchard)

- Pedersen commitment (or something very similar) for value commitments.
- Currently, use Zcash's Pasta Curves

Future

Current stage of Taiga is many details still to be confirmed, the design is solid. The Future of Taiga is bright. Eventually we are planning to unify all the execution environments into a single one, which is long term plans.

FHE is complicated and probably want to have a closer integration with Ferveo, so we can prove correspondence b/w private state and the data encrypted to Ferveo using ZKPs. Ferveo is for encrypting Txs

, so you couldn't order them in a sneaky way so if we could prove correspondence between private state (FHE) and data encrypted to Ferveo.