This is an initial draft of research proposal of private solving using MPC, which contains a summary of my understanding of the setup together with the potential directions I consider. @cwgoes, @vveiln, @apriori, @degregat I seek your feedback to correct/widen my understanding if it is mistaken or incomplete and to settle down the problem formulation. Please let me know if the scope seems to be trivial or infeasible. Questions are italicized to prevent being missed. I understand that answering some of them could be already a part of this project - I just want to be sure that they are to-be-answered in that case.

@cwgoes - Based on the feedback I get for this, I might write a better-organized version for Mary -and potentially add more context for some parts-

I start with a quick list of two possible research directions since I will refer to them while introducing the setup. Details on them will follow.

- Generic MPC: Compatible with any algorithm. Provides flexibility, comes with practicality concerns.
- Specialized MPC: MPC protocols tailored for specific problems. Limited/no flexibility on algorithm choice, advantageous for efficiency.

What is intent?

As far as I understand from @degregat, intents are implemented as imbalanced transactions. After solving is completed, new transaction needs to be computed anyway. So, I think I can consider intents as an abstract entity, which will be encoded in whatever way is convenient for MPC. Intents essentially contain want and have fields which include the kind and quantity. There are also different representations of intents in kudos-snippets github repo, I am not sure how general the intents I consider should be. Considering that end-goal is a PoC implementation for Kudos, it might be best to catch up with what is the requirement there.

My suggestion to scope intents for this project would be: intents contain have.kind, want.kind, rate, max_quantity, where rate is the desired amount of want.kind resource for 1 have.kind resource and max_quantity is the maximum amount to trade have.kind resource. Any solving strategy that trades have.kind and want.kind for the desired rate up to max_quantity is valid.

I guess this view of intent is already practically useful and it is restricted enough to leverage cycle detection/flow maximization kind of algorithms. This is especially useful if we will investigate specialized MPC protocols rather than generic frameworks. What do you think about this scoping of intents?

There seem to be some weights in Kudos Juvix specification, should I consider weights too?

If so, I need to understand how weights are expected to work. Who is a source of up-to-date information on this?

What is solving?

[

solving intents

1916×1067 111 KB

](https://europe1.discourse-cdn.com/standard20/uploads/anoma1/original/1X/025ae921fa52ae660054b96b0c226faf00f84e14.jpeg)

In this context, I use the term solving to refer to the process of:

1. Collecting intents from the users and matching a suitable subset of them: I see two questions here. First one is the algorithm picked for matching, while the second one is running this (or an) algorithm without revealing intents of the users. More details below.
2. Creating a (possibly shielded) transaction: To my understanding, this is what Taiga handles. Does this imply that we have to stick with generic MPC constructions at least for transaction creation? Can Juvix be used to convert Taiga code to circuits?

In that case, we might be able to use some of existing frameworks on Taiga without too much modification.

How to match (without the question of privacy)?

Answer to this question heavily depends on the definition of intents and valid match. I'll state some generic discussion.

I have found topic Intent Languages: CSP Formulation where CSP formulation of intent solving is discussed and a solution based on integer programming is adapted. I think the discussion there is useful but the generality of CSPs seems to be an obstacle to coming up with a tailored, efficient solution. Investigating whether certain CSP classes are known to solve with MPC can be an option. Here, I am not sure if decision or maximization (checking vs. solving discussion) version is more relevant. With MPC, we can either require some subset of participants to be matched (which can be all participants, too) or all participants to be matched. We need optimization version for the former, but both optimization and decision versions work

for the latter.

The problem can be also viewed as a flow maximization problem. If we assume that there are too many kinds of resources to be matched (sounds reasonable for Kudos), an initial step to match intents could be matching want-have kinds and postponing the numerical matching. For this, cycle detection algorithms can be useful (which seems to be a separately studied problem in MPC framework). If we expect to have a small set for possible kinds of resources that an intent can contain, then this might hurt the privacy, in addition to being unnecessary.

How to solve privately:

Some candidate approaches for private solving are discussed in this report and blog post. For this project, I will focus on the MPC approach. With MPC, we aim to prevent leakage of users' intents. In this setting, users jointly play the role of solver without sending their intents explicitly. I see two potential scenarios to consider:

- MPC participants only consist of the users. Users jointly match the intents and settle the transaction.
- There is an additional MPC participant, who is responsible for handling heavy computations while users are mostly responsible for exchanging easy-to-compute messages. This might be useful to consider if an MPC protocol for the above item requires too much computational resources for users. (I don't even know if this kind of construction exists.)

As a counter-argument to employing the second approach, there seems to be a recent MPC paper (with open-source implementation) that is -claimed-to-be- scalable and optimized for lightweight participants. Checking this specific work in detail may be a good start to the project.

Matching in MPC could mean both matching intents of all the participants (i.e. decide whether participating users could all match, and then create the transaction if so) and matching intents of a subset of the participants. I believe both are valuable and I consider focusing on the one that seems more promising on the way. Does this sound okay?

Either of them can be favorable based on topology assumptions (which I believe we don't have).

As @vveiln already pointed out, there are some obstacles that we may need to keep in mind:

- Since users are not unconditionally trusted, a user can attack privacy of another user by crafting intents. This is proposed as a reason to consider more than 2 parties in the computation. Isn't it also possible that all of n-1 other users engaging in joint solving are malicious and colluding for any n? It might be possible to argue that for some big enough n, this is as unlikely as we can accept. What is the reasonable definition of big enough n here? Is considering 3-party computation worth the privacy gain in exchange for lost efficiency? I guess answering this question requires one input from network side -the rate of increase in privacy with n- and another one from MPC side -the rate of efficiency decay with n-.
- A follow-up question to ask is "what should be minimum number of intents that are matched to settle a transaction". Even though many parties participate in the solving protocol, similar attacks could be possible if it is possible to match just 2 intents.

Do we have some answers/preference on the answers to these obstacles or should they be kept as side-questions in the project?

Existing PoC implementations for solving research:

- kudos-snippets github repo: Stand-alone (i.e. no integration with Taiga etc./transparent case only) demonstration of the general idea of intents and solving using a CSP solver.
- mpc-solver-approx github repo: Stand-alone (i.e. no integration with Taiga etc./transparent case only) demonstration of a basic solving strategy using MPC framework MP-SPDZ.

Scope proposal:

As far as I understand, implementation of solving with MPC for some specified algorithm is not the big question. Although there is no standard MPC framework yet (apparently NIST is engaging in some standardization process soon - but definitely not the solution for us in near future), there are available frameworks that can run any algorithm that we specify with their language. MP-SPDZ (github, paper) is the one @vveiln used for PoC already and awesome-mpc has an extensive list of frameworks that we can check. The main problem seems like the practicality of implementation.

Potential directions to focus on here are:

1 - Conducting literature research to list generic frameworks/MPC protocols with advantages and disadvantages. Benchmarking a chosen solving algorithm on different MPC protocols using a comprehensive framework like MP-SPDZ. This can include sketching trade-offs between different efficiency measures like number of rounds, size of messages, total computation time, and local resources used.

Potential (rough) agenda for this: Collecting advantages/disadvantages of existing theoretical/implemented generic constructions/frameworks. Picking a framework for benchmarking and learning how to use it. Running experiments for

benchmarking and reporting them.

Struggle point: Whatever generic MPC framework we use, it will not be useful if it is not compatible with Taiga. Is the correct question to ask here finding a framework that is compatible with Taiga or finding a good protocol and implementing it in a way compatible with Taiga (I don't expect this to be something I can achieve within the scope of this project)?

2 - Trying to optimize MPC and solving algorithms together. This approach includes iterating over different solving strategies by checking specialized MPC protocols and/or trying to develop nice solving strategies using the problems that can be efficiently solved with MPC. Potential problem here: What about after matching is done? Although we specifically handle the matching part in MPC, do we have to go back to some generic MPC for transaction creation?

Potential (rough) agenda for this: Work on the protocol+algorithm design. PoC implementation for the chosen design. Possibly choosing a few designs and experimenting with the tradeoffs. Reporting.

Which one of these two directions is good/better to pick? Or am I completely off-track?