

Analysis

Project Outline

This is a project aimed at helping groups of people manage money, secured with digitally signed transactions. It also provides a way to quickly and easily settle chains of debt [\[1\]](#). To interact with the final product, a simple, easy to use command line interface will be provided.

Features

- Cryptographically signed transactions guaranteeing security and integrity of your transactions
- Simplify chains of debt in your group
- Command Line Interface (CLI)
- Client / Server Model
- Database

Non-Features

- None of the user's money is ever put into the app. This is simply a tracker, you cannot settle debts through the app
- No policing of people who do not pay their debt - this is a problem for people in the group to deal with as they choose

Background to the Problem

A common problem for many young people is that of money. Specifically, keeping track of who owes who how much money in a group of friends. Arguments about how much money is owed are commonplace. This is something I often see amongst my own group of friends. I know one person in particular ([the end user](#)) feels as though he is never paid back, and would like to see a solution to the money tracking problem.

In order to arrive at a solution, what is needed is a reliable, trustworthy way to track money. People who use the tracker will need some sort of guarantee that people cannot 'hack' the app, changing people's debts. Since I am the creator, and likely a future user of this app, my friends also need confidence that I will not be able to write off all of my debts. Hence, that is problem number 1 - **secured transactions**.

A problem inherent to a money tracker is that it is just that - a tracker. Since no money flows through the software, long chains of debt may form. This may result in people needing to make many small transactions to different people in order to pay what they owe. This seems a shame, when it is possible to have the software simplify the debts into a minimal number

of transactions per person, to ensure money flows as efficiently as possible through the network. This idea was presented to me by the end user, but I anticipate it to be an integral part of the project, hence the brief mention here. This makes problem number 2 - **efficient settling of group debt**.

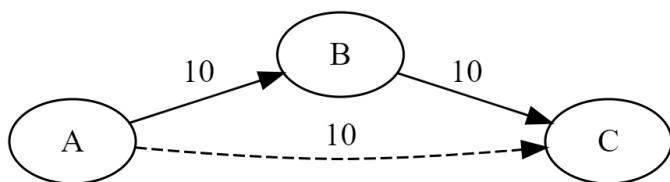
The End User

The end user that I had in mind is the friend who inspired this project. I think he fits perfectly within the target market - 13-18 years old, fairly sociable, not too technologically minded, and just wants an easy way to keep track of his transactions in his group of friends.

My interviewee was keen to point out that people may abuse this system. They may just use it to rack up debt, and then never pay anyone back. He wanted to know if there was a way that I could stop this occurring. To this I replied no, not really. They can do the same thing without the app. It is your decision whether to lend to them. With the app however you can see exactly how much they've taken - it provides indisputable accountability. It does however assume that people are willing to pay up. It is up to the user to deal with the eventuality that they don't.

Another problem that he identified was that of chains of debt. He (rightly) pointed out that if you owe people who owe people, it's sometimes easier to cut out the middle man and turn two discreet transactions into one smaller one. This idea naturally extends to a group of friends, who may all have varying levels of debt between them. Thus, instead of making the group do a large number of transactions with money going back and forth frequently between the same hands, I will aim to let a group settle in the easiest way possible.

A valid concern with this plan is the fact that some may end up owing people they didn't before the simplification. Consider a simple case where A owes B £10, and B owes C £10. Two transactions could be reduced to 1 if A were to pay C directly. However, in a larger group, people may not like giving money to people who they do not directly owe on the will of my program. Hence, an important constraint is that no one owes someone that they didn't owe before settling occurred (see image below.)



Even though the dashed edge would reduce transactions, it should not be added.

The end user suggested that I keep the interface as simple as possible. He maintained that he didn't want lots of unnecessary frills - just a simple, functional interface. To this, I suggested the use of a CLI. I was a little concerned that most people would not have used one before and may not know what it is. However, once I explained the concept, he seemed to come round to the idea. Its main benefit over a graphical user interface (GUI) is that it is unambiguous. It is also, arguably, easier to do things with a CLI once you become comfortable using one.

The final main worry that my interviewee brought up was that of guaranteed security. I had talked to him when I had the idea for this project, before I had learned about asymmetric encryption and cryptographic signatures. He wanted to know how a transaction coming from him could be verified as his, and no one else could pretend that they are, say, owed lots of money. He also didn't trust me, and said that if our group of friends started using this product, he would suspect that I would "code away my debts".

In short, the problems identified here are as follows:

- How to make sure that users trust the integrity of the transactions, making sure that users know that no one can tamper with their debts.
- How to settle debt across large graphs efficiently (here meaning few transactions per person)
- How to make a CLI that is as simple as possible

This is not an exhaustive list - it leaves out all the technical problems I will likely face, which are discussed in the next two subsections

Research of existing solutions

Currently, on the market, there are a few products similar to that which I am proposing. Having surveyed a few options, I decided to look at Evenfy and Splitwise in more detail.

Both work on the same premise that I have outlined: an intuitive way to track who owes who in a group.

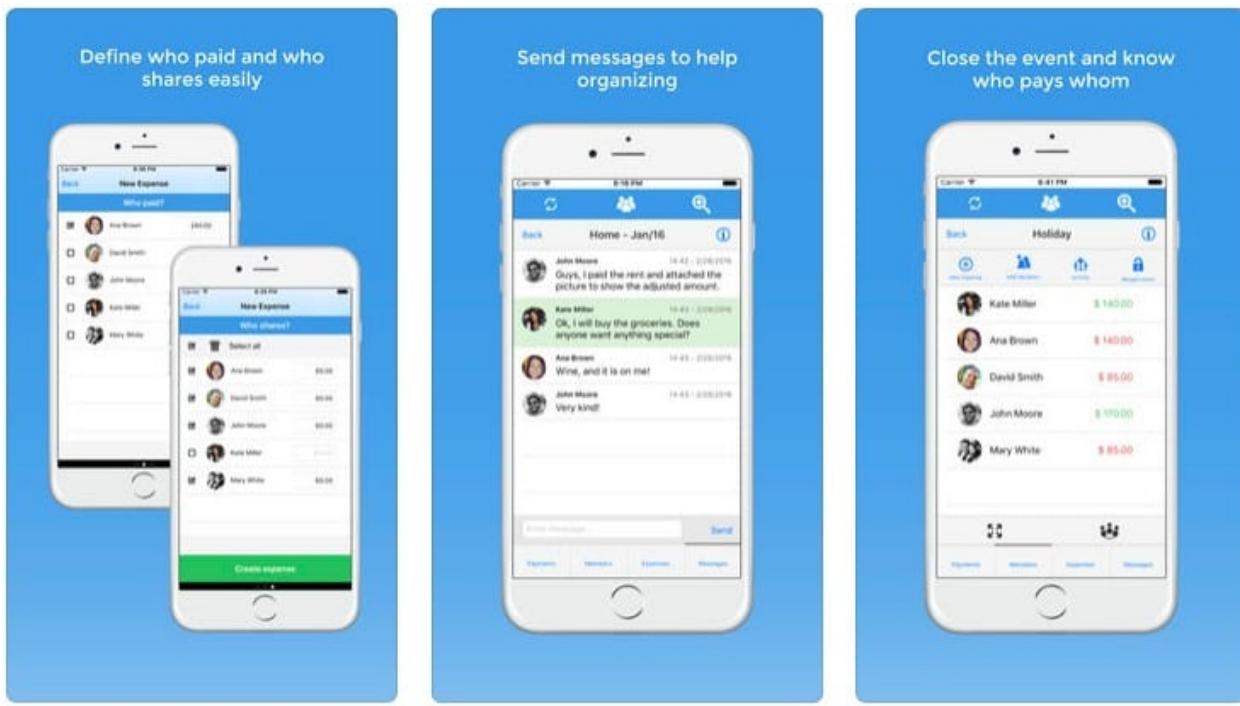
This is all accurate at time of writing.

Evenfy

Evenfy is an app that does exactly what I set out to achieve. It mainly focuses on group expenses, and can be accessed from a computer.

It has an interesting feature in that it allows for temporary groups to be created in order to track short term expenses, such as over the course of an event. Evenfy tries to learn about common expenses, and suggests who pays over time. This may be useful if you rent a house with a few others, is what Evenfy say.

Evenfy will also calculate the easiest way to settle the group; that is, ensure that everyone's debt goes to 0. This is an integral part of keeping an expense tracking app usable in my opinion, and in the app's reviews, users seem to agree.

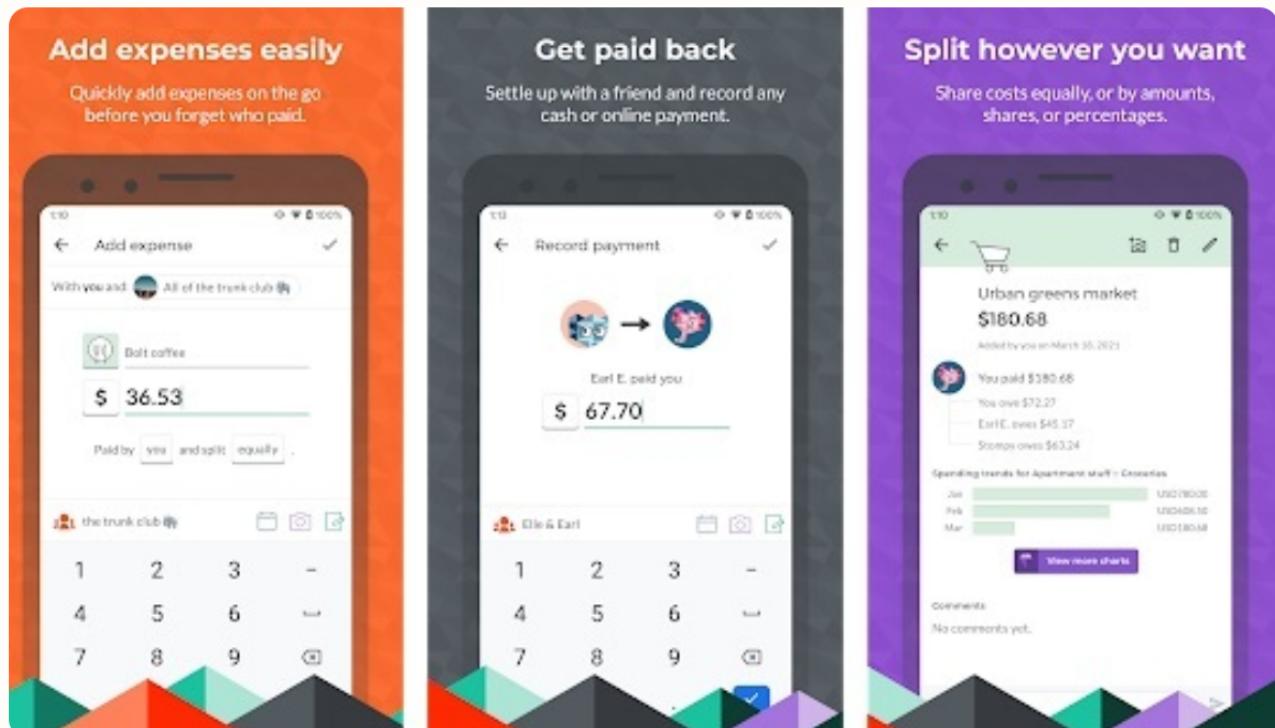


Evenfy also allows for a group to settle debts easily. This feature will be discussed more in the next product evaluation, as an identical feature appears there.

Evenfy is free to use for the first 6 months, but then requires monthly subscription of 99 cents if you want to track more than 10 expenses per month.

Splitwise

Splitwise is similar to Evenfy in many ways. It allows the easy splitting of bills (by percentage or equally), and keeps track of who owes whom within the group.



After an account is created, you can create a group of people. Splitwise will then track all expenses in this group. Expenses can be referenced, and you can see at a glance exactly how much you owe.

In comparison with Evenfy, the overall experience and feature set is similar. Both are well put together and include features that I think are unnecessary for my target market. I do prefer Splitwise's UI slightly: it is easier on the eyes, and slightly less cluttered. Both have an excellent UX.

My only complaint is that I feel as though I have to search through a user interface for many quite simple tasks. The settings menu in particular feels like it obfuscates settings such as deleting your account, as well as updating user information. This is, however, common to many graphical user interfaces. As I do not plan on building a graphical interface, I do not think this is something that I need to be too mindful of. Instead, I will strive to make the CLI as straightforward as possible.

In terms of flaws, Splitwise requires a £2.99 per month subscription to unlock every feature (most core features are free to use). This is, in my opinion, better than Evenfy's payment model. However, the point is moot as I will not be charging for the use of this project.

The interesting part of these apps is in the debt simplification process. Since neither are open source, one cannot know for sure how the debt simplification is done. However, after much investigation, I found a few possible options.

Givers and Receivers

This, I believe, is the less likely of the two possible approaches, because it reduces down to a decision theory problem which is NP-Complete. It also takes a few passes of the data to get there. It is not particularly efficient. The steps are as follows

1. Calculate the net flow of each node
2. Categorize nodes into 'givers' (those who overall owe money) and 'receivers' (those who are overall owed), and those who owe/are owed nothing.
3. Settle any '1-1 transactions', where one 'giver' can completely remunerate a 'receiver'.
4. Settle any transactions where a receiver is owed a perfect subset of givers money (i.e. a receiver who is owed £5 could be settled by two givers, with £3 and £2)
5. Settle any remaining transactions by splitting money from givers in a greedy fashion to receivers.

The 'NP-completeness' of the problem starts at stage 4, when lots of computation has already been done on the data.

Stage 1 reducing all the transaction from Person A → Person B to a single number. This is then done for the whole group, and a weighted digraph is built, where edges represent money owed. A residual graph, wherein an edge in the opposite direction with a weight $\times = -1$ the initial weight is added.

A breadth-first-search, where each edge traversed from the current node (u) has its weight recorded (and summed when all edges from u are explored) is required to obtain the flow of the graph. This gives a time complexity of $\mathcal{O}(|V| + |E|)$.

The second step can be done very efficiently, in $\mathcal{O}(\log V)$ time if a mergesort is used to sort the nodes in descending order of money they have (assuming those owed have negative amounts of money). Then the list of nodes can be split into two subsets at the point where 0 would be inserted into the list (those with a net debt of 0 can be removed before splitting).

Step 3 could be done by walking through the sorted givers list. For each node g in the givers list, you would walk down the receivers list (r in receivers), settling any transactions where $g[money] = r[money]$. (Any nodes which are not 'filtered' in this way move to a new stage of processing). This would give a time complexity of $\mathcal{O}(G \cdot R)$, where G is the number of 'givers,' and R is the number of 'receivers'. This can be made much more efficient by using sorted lists, and remembering how far the receivers list was walked down in previous runs.

In pseudocode

```
for giver in givers:  
    for receiver in receivers:  
        if giver.money > receiver.money:  
            // giver has more money than any receiver in receivers  
            // receiver passes filter  
        else if giver.money < receiver.money:  
            // receiver is owed more money than any giver in givers has  
            // receiver passes filter  
        else:  
            // giver and receiver have the same amount of money so a 1-  
            // can happen  
            settle(giver, receiver)
```

Step 4 becomes a sum of subsets problem, and is NP-Complete, i.e. there is no algorithm which can solve this problem in quicker than exponential time. It is at this stage that this approach becomes infeasible for the real world, and thus, A better heuristic must be considered.

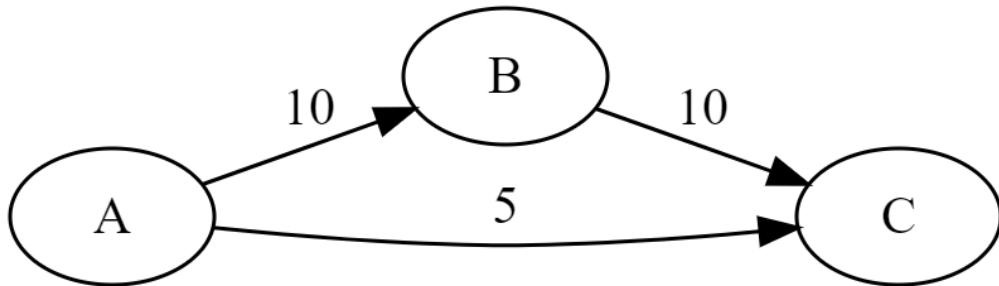
Another problem with this approach is that it contradicts one of my initial high level requirements. It does not preserve any sense of who owes whom past step 1. Thus, it cannot guarantee that no one will owe anyone that they did not previously owe.

For these two reasons, I shall elect to not use this method.

Max Flow

I think that this problem can be modelled as a problem of flow. The question 'how can we minimise the number of transactions one person has' can be reduced to 'how do I maximise the amount of money I send to one person'. If the amount of money sent to someone is maximised such that no one ever has to pay more than they owe, then people will, on average, have fewer transactions to make after settling.

Consider the simple case of three people, Alice, Bob and Charlie. Let Alice owe Bob £10, Bob owe Charlie £10, and Alice owe Charlie £5. This can be represented as a weighted digraph



Notice that there is a chain from $A \rightarrow B \rightarrow C$.

This chain shows that all £15 from Alice will eventually end up with Charlie, even though some of it goes via Bob. Thus, this can be simplified by cutting out the middle man, and instead letting Alice owe Charlie £15.

I believe that this functionality can be achieved through a slightly unusual application of the Edmonds-Karp Max Flow Algorithm.

Fulkerson-Ford Max Flow

The Edmonds-Karp Max flow algorithm is really a combination of two separate algorithms: a breadth first search, and the Ford-Fulkerson max flow algorithm.

Ford-Fulkerson aims to answer the question: how much flow can one push along a network, without exceeding the capacity of any edge? The algorithm works on flow graphs.

The way in which the algorithm works is simple:

- 1) Find an augmenting path from source node to sink node (through the residual graph)
- 2) Augment flow down path
- 3) Repeat until no more augmenting paths exist

To define some terms:

Flow Graph

A flow graph is a form of weighted digraph, in which edges have a flow and a capacity (as opposed to a single weight). Each edge has the notion of remaining capacity (remaining capacity := capacity - flow). Each edge is initialised with flow = 0. This value changes during the course of the algorithm.

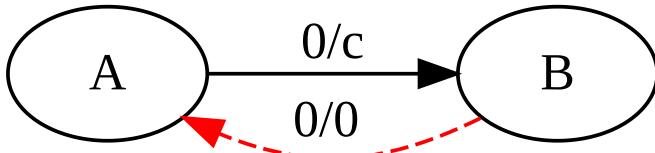
The flow of an edge is never allowed to exceed its capacity. (i.e. edge will never have a negative remaining capacity).

Augmenting Path

The augmenting path is a path of edges in the residual graph, where each edge has a remaining capacity > 0 . The path is from two specified nodes - a source node s and a sink node t .

Residual Graph and Residual Edges

The residual graph is the combination of the flow graph , and residual edges. For each original edge from $u \rightarrow v$, with capacity c , there exists a residual edge from $v \rightarrow u$, with capacity 0.

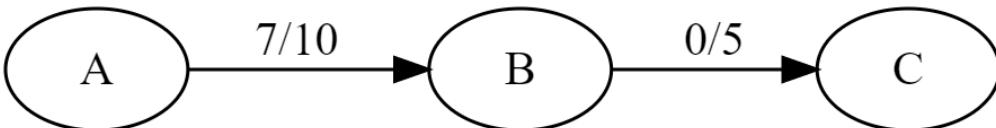


Residual edges are valid edges to consider when looking for an augmenting path, given that they have unused capacity (the above example's residual edge has an unused capacity 0, as $0 - 0 = 0$).

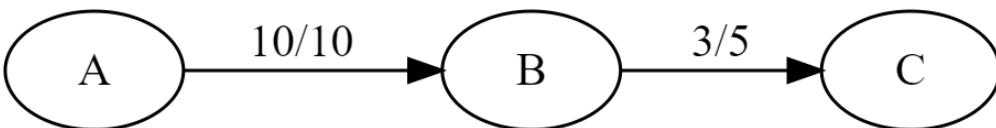
Augmenting the flow

The act of pushing as much flow as possible along an augmenting path.

The amount of flow pushed down the path is equal to the bottleneck value of the path. The bottleneck value is given by the edge with the smallest amount of unused capacity



In the above case, the bottleneck value is 3. This is because $A \rightarrow B$ has 3 units of unused capacity, and $B \rightarrow C$ has 5 units of remaining capacity. Thus, the maximum amount of flow that can be pushed through this augmenting path is 3 units. After augmenting the flow, the path looks like this



When flow is augmented, it is important to keep the net flow through the node the same. This is done through the residual edges, which are updated so that they have a flow of $-c$. In this example, the residual edge from $C \rightarrow B$ would have a capacity of -3 .

An example of the algorithm working is provided in the next subsection

Once no more augmenting paths can be found, the bottleneck values of each of the augmenting paths are summed. This value is the max flow through the given graph from the source node to the sink node.

Complexity

Finding an augmenting path is completed in $\mathcal{O}(E)$ time (where E is the number of edges in the graph). In the worst case, 1 unit of flow is added every iteration. This makes the overall time complexity of the Fulkerson-Ford Max Flow $\mathcal{O}(E \cdot f)$, where f is the max flow of the graph.

This is not ideal, as the time complexity is heavily dependent on the flow through the graph. This is improved upon in the strongly polynomial Edmonds-Karp Max Flow algorithm.

Edmonds-Karp Max Flow

Edmonds-Karp Max Flow differs from Fulkerson-Ford in the finding of augmenting paths. Fulkerson-Ford does not specify how an augmenting path should be found, whereas Edmonds-Karp finds the shortest [2] augmenting path from s to t .

This is ensured by using a Breadth First Search (BFS) to find augmenting paths.

A short augmenting path is favourable, as the longer the augmenting path is, the higher the chance of an edge with very little unused capacity. This could lead to edges reaching capacity in more iterations, giving a considerably slower runtime. As aforementioned, the worst case is that every path has a bottleneck of 1 unit of flow. Since Edmonds-Karp uses a BFS to find augmenting paths, we are guaranteed the shortest (in terms of number of edges traversed) path from s to t .

This detail gives Edmonds-Karp a much more favourable time complexity, of $\mathcal{O}(EV^2)$. This is a product of the time complexity of a BFS, $\mathcal{O}(V^2)$ (when using an adjacency matrix to represent the graph) and the Fulkerson Ford time complexity, $\mathcal{O}(E \cdot f)$. However, flow does not appear in the time complexity of Edmonds-Karp.

A property of BFS is that, when it finds a path from a source node s to a target node t , that path is guaranteed to be the shortest path (P_n) from s to t . A corollary of this is that P_{n+1} is guaranteed to be a longer path than P_n . This reduces the upper bound run time of one iteration of Edmonds-Karp to $\mathcal{O}(E)$ versus the original $\mathcal{O}(E \cdot f)$. A more complete proof of time complexity is provided here [3].

Since Edmonds-Karp's runtime is independent of flow, its input, it is classed as a strongly polynomial algorithm, making it perform much better than the Fulkerson-Ford max flow algorithm. Thus, this is the algorithm that I will be implementing to simplify debts across a group.

Settling a graph using a Max Flow algorithm

Having explored various max flow algorithms, the question now becomes how to settle an entire graph's worth of debt. This is a fairly challenging problem since max flow algorithms only work on a source node and a sink node.

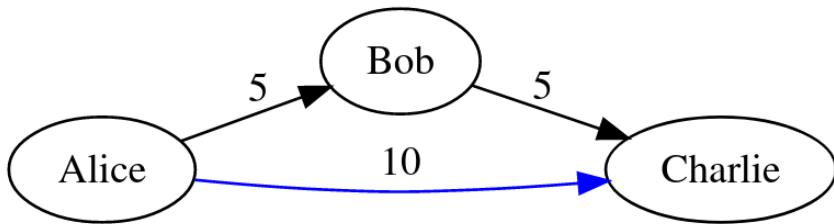
After researching, I found a solution which proposed the following.

```

// initial graph is the initial network of debts

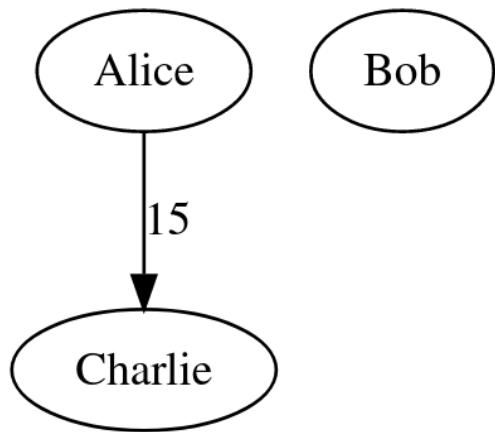
clean_graph = WeightedDigraph()
for edge(u, v) in initial_graph:
    if flow := max_flow(u, v):
        // append an edge to the new graph from u -> v with weight flow if
flow > 0
        clean_graph.append(u, v, flow)
        // remove edge that has been maxflowed
        initial_graph.remove_edge(u, v)

```



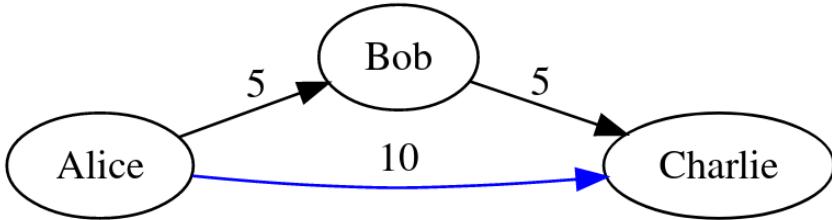
In prose, a new weighted digraph is generated (with no edges, but all the same nodes) for the cleaned edges.

Starting on node G , we would therefore run a max flow from $G \rightarrow B$. If the max flow from $G \rightarrow B > 0$, then an edge with the weight of the max flow from $G \rightarrow B$ is added to the new weighted digraph. The edge from $G \rightarrow B$ in the flow graph is deleted. This process will happen again from $G \rightarrow D$. After having explored all neighbours, the BFS continues, until every edge in the graph has been settled. In the above example, a valid settling could look like this [4]



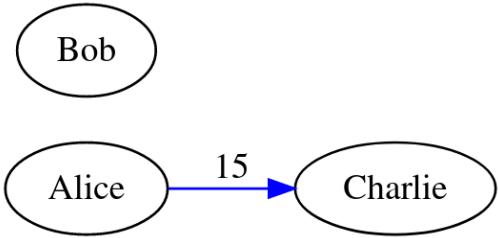
However, while hand tracing the aforementioned algorithm, I discovered that it was not a correct algorithm.

Take the original, unsettled graph, and select the edge in blue first

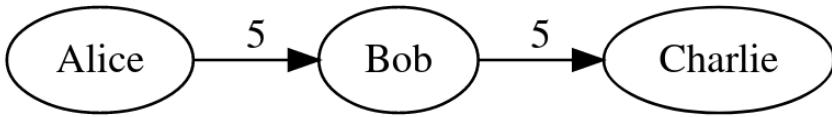


Running Edmonds-Karp along this network with Alice as a source and Charlie as a sink gives a max-flow of 15. Thus, add an edge to the new graph from Alice to Charlie with a weight of 15, and remove the Alice - Charlie edge from the initial graph

This gives the new graph:



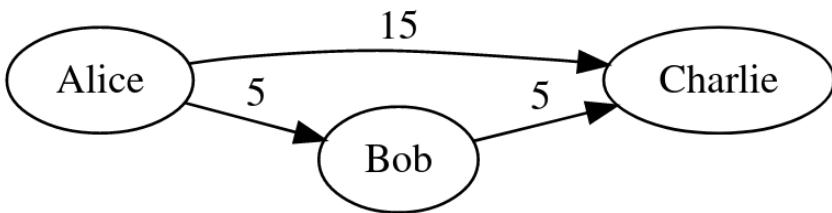
And the 'initial' graph



Repeating this process for the remaining two edges gives us max-flows of
Alice → Bob: 5

Bob → Charlie: 5 (after removing Alice → Bob)

The 'clean' graph that is generated will therefore look like this



Notice that initially, Alice owed the group £15. Now she owes the group £20. Similarly, Charlie was owed a total of £15 pounds by the group, and is now owed £20.

Thus, this algorithm is incorrect.

The reason why is that it does not account for how max-flow is generated. In the case of the first edge we just considered, we calculated a max-flow of 15. This was achieved by pushing 5 units of flow down the Alice → Bob → Charlie path, and 10 units of flow directly from Alice → Charlie.

Since all of these paths become saturated, it should be the case that **no more flow can be pushed through the graph**. This algorithm only removes the considered edge, instead of all the edges saturated by the max-flow algorithm.

The approach I decided to take was to modify the initial graph in place. After a max-flow is run, and a new edge is added to the clean graph, the 'initial' graph is restructured. The restructuring replaces the capacity of each edge (u, v) to its original unused capacity. Like this, any saturated edges are removed, and future iterations of the graph are constrained to only produce results based on outcomes of previous iterations.

You may assume that this step is unnecessary, as it should be the case that the max-flow algorithm does not consider saturated paths. However, the max-flow runs on the residual graph, and thus may augment flow in a way that 'removes' money from a previous transaction. Since the transactions are solidified in another graph, this creates a discrepancy, and was where the error in the algorithm I found online lay.

Hence, I implemented the algorithm as follows:

```
clean = FlowGraph()

for edge(u, v) in graph:
    if flow := max_flow(u, v):
        // append an edge to the new graph from u -> v with weight = flow
        clean.add_edge(u, v, flow)

        // restructures edges as described above
        graph.reduce_edges()

    // for loop now runs on until no more edges in graph;
    // thus stops when no edges left in initial graph
```

In the worst case scenario, only 1 edge is removed from the graph each time

`graph.reduce_edges()` is called. In this case, 1 max-flow would happen per edge of the graph, giving a time complexity of $\mathcal{O}((EV^2) \cdot E) = \mathcal{O}((EV)^2)$. However in practice, more than one edge will become saturated with each max-flow, reducing the expected time complexity.

Note: **Implications to security**

Since the server that is settling a group of transactions is creating and destroying new transactions that haven't happened in the real world, the signatures that transactions were initialised with will no longer be valid after settling.

Thus, the user should resign any transactions in the group that they are involved in after the settling has happened. This allows the solution to remain secure, and also lets the user see exactly what has happened to their debt.

High Level Objectives for the Solution

After careful consideration of the end user, and existing systems, I can arrive at my high level and low level requirements

On a high level, my objectives are as follows:

1. An RSA implementation that will allow the signing, and verification, of transactions
2. A way to settle the debts of the group in as few as possible (heuristically speaking) monetary transfers
3. A server-side component of the application which can verify transactions, and store / retrieve them from a database
4. A client-side component of the application that will have a simple user interface (CLI)
5. A database that should be able to store user and transaction information

Low Level Requirements

For the purposes of testing, these are low level requirements that I would like to fulfil. The first three sections are very low level, and as such may be hard to understand in context of the project. Section D paints a picture of how achieving my aims in sections A, B and C will apply to my project.

RSA Implementation (A)

1. A reliable interface to a hashing module
2. RSA Key Handling:
 1. Be able to load RSA public/private keys in PEM format from files / STDIN
 2. Be able to validate the format of these keys
 3. Be able to parse these keys extracting all necessary numbers for RSA decryption
3. Signing/Verification
 1. Have a valid RSA encryption scheme (encryption with public key)
 2. Have a valid RSA decryption scheme (decryption with private key)
 3. Have a valid RSA signing (sig) scheme (signing with private key)
 4. Have a valid RSA signature verification (verif) scheme (verify with public key)
4. Object Signing
 1. Algorithm to convert an object to a hash in a reproducible way, minimising the chance of hash collisions
 2. Ability to sign a class of object with RSA sig scheme
 3. Ability to verify a signed object with RSA verif scheme, raising an error if signature is invalid

Debt Simplification (B)

1. A reliable digraph structure, with operations to `transactions.graph.GenericDigraph`

1. Get the nodes in the graph `nodes()`
2. Check if an edge exists between two nodes
3. Nodes can be added
4. Nodes can be removed
5. Edges can be added
6. Edges can be removed
7. Neighbours of a node should be easily accessed (neighbours for the purposes of a breadth first search)

2. A reliable flow graph structure

1. All the operations listed in B.1.1
2. Adding an edge should have different functionality: edge should be able to be added with a capacity, and edges should have a notion of flow and unused capacity
3. Be able to return neighbours of nodes in the residual graph (i.e. edges, including residual edges, that have unused capacity)
4. A way to get the bottleneck value of a path, given a path of nodes

3. A reliable recursive BFS that works on flow graphs

4. Implementation of Edmonds-Karp

1. Way to find the shortest augmenting path between two nodes
 2. Way to find bottleneck value of a path
 3. Finding max flow along a flow graph from source node to sink node
5. Simplifying an entire graph using Edmonds Karp, using the method laid out in [Settling a graph using a Max Flow algorithm](#).
6. Be able to convert a list of valid transactions into a flow graph
 7. Be able to convert a flow graph into a list of transactions, signed by the server
 8. Be able to simplify a group of transactions, having each transaction individually verified before settling

Client / Server Structure (C)

1. The server should be accessible to the client via a REST API
2. The client should be relatively thin, only dealing with input from user and handling error 400 and 500 codes gracefully.
3. Client and Server should communicate over HTTP, using JSON as an information interchange format
4. The client should have a clear, easy to use command line interface

'Integrated' requirements for how the end system should behave (D)

1. Ensuring the validity of transactions

1. If a transaction is tampered with in the database, it should be classed as unverified
2. A user should not be able to sign an already signed transaction
3. A user should not be able to sign a transaction where they are not one of the listed members
4. A user should not be able to sign a transaction with a key that is not associated to their account
5. A user should not be able to sign a transaction without entering their password correctly
6. Every time a transaction is pulled from the database and sent to the user, it should be verified by the server using the RSA sig/verif scheme from section A

2. Ensuring that the debt simplification feature works

1. All transactions in the group being settled should be verified upon being pulled from the database
2. It should not be possible to simplify a group if there are unverified transactions in the group
3. If the transaction structure of the group does not change, the user should be notified
4. The simplification should accurately simplify a system of debts such that no one is owed / owes a different amount of money after simplification
5. The simplifying process should result in unverified transactions being produced, able to be signed by the user

3. Ancillary features

1. Users should be able to register for an account, providing name, email, password and a PEM formatted private key
2. Users should be able to create transactions where they are the party owing money; these transactions should be created as unsigned
3. Users should be able to create a group with a name and password
4. Users should be able to join a group by group ID
5. Users should be able to mark a transaction as settled; transactions should only be marked as settled when both parties involved mark the transaction as settled
6. Users should be able to see which groups they are a member of
7. Users should be able to see all of their open transactions
8. Users should be able to see all the open transactions in a group (whether they are part of the group)
9. Users should be able to see the public key information of any user on the system
10. Users should be able to see individual transactions by passing in a transaction ID

Database Architecture (E)

1. User information

1. User ID
 2. Contact info
 3. A hash of the user's password
 4. Associated Groups
 5. Public Key (provisions for one or more)
2. Transaction Information
 1. Transaction ID
 2. Payee
 3. Recipient
 4. Transaction reference
 5. Amount (£)
 6. Payee's signature
 7. Recipient's signature
 8. Whether transaction has been settled
 3. Group information
 1. Group name
 2. Group password
 3. People in the group
 4. Transactions in the group

Project Critical Path

The order in which I will carry out the project in 6 distinct phases

1. Cryptography
2. Debt Simplification
3. Combination of 1 & 2 into a fully encompassing transaction object
4. Database setup and design of SQL statements to retrieve data
5. An API designed to allow communication between client and server
6. User Command Line Interface design

In more detail (Key: Black boxes with coloured text are labels)



Design

High Level System Design

Briefly, my goal for the system is to be able to create transactions, and digitally sign / verify them. Then, the system should be able to simplify chains of debt among people.

The notion of signing and verifying means that the project will require a sizeable cryptography aspect, especially as is my intention to write the RSA encryption / decryption myself.

I will also need a mechanism with which to be able to simplify large groups of debt. As discussed in the analysis section, this will be done through the use of flow graphs.

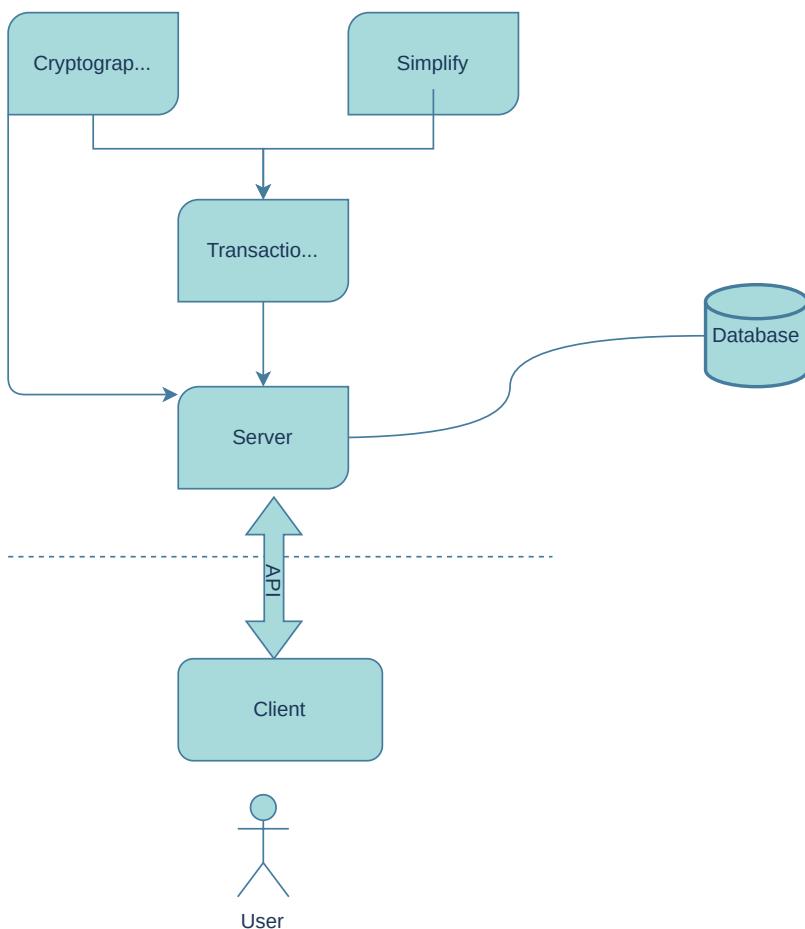
Next comes the transactions themselves. Only at the transaction level should the concepts of cryptography or the graph processing start being used.

Finally, I will assemble everything together at the server level, although the simplification module should never need to be directly used. The cryptography module will require light

use for the hashing of passwords.

To store these transactions, along with user data and supporting infrastructure, a fairly complex database system will be employed.

As a diagram



Note: the client needs a way to generate hashes of passwords. I intend to use the cryptography module that I will write. This is not necessary however, as any SHA3_256 hash will be sufficient.

On a high level, the system is decomposed into 5 separate modules, each accounting for a main component of the system (as above). These are `crypto`, `simplify`, `transactions`, `client` and `server`. Three of these modules, `crypto`, `simplify`, and `transactions` will all have a folder of unit tests associated with them.

The modules in the final design should be as decoupled as possible. `crypto` and `simplify` should not have any dependencies on any other modules (as will become apparent in class diagrams above)

Expected File Structure

```
.
├── README.md
├── requirements.txt
└── settle_db.sqlite
```

```
├── setup.py
└── src
    ├── client
    │   ├── client.py
    │   ├── cli_helpers.py
    │   ├── cli.py
    │   └── __init__.py
    ├── crypto
    │   ├── hashes.py
    │   ├── __init__.py
    │   ├── keys.py
    │   ├── rsa.py
    │   └── sample_keys
    │       ├── d_private-key.pem
    │       ├── d_public-key.pe
    │       ├── m_private-key.pem
    │       ├── m_public-key.pe
    │       ├── t_private-key.pem
    │       └── t_public-key.pe
    ├── __init__.py
    ├── server
    │   ├── endpoint.py
    │   ├── __init__.py
    │   ├── log
    │   ├── models.py
    │   ├── processes.py
    │   ├── resources.py
    │   ├── schemas.py
    │   └── setup.py
    ├── simplify
    │   ├── base_graph.py
    │   ├── flow_algorithms.py
    │   ├── flow_graph.py
    │   ├── graph_objects.py
    │   ├── __init__.py
    │   ├── path.py
    │   └── weighted_digraph.py
    └── transactions
        ├── __init__.py
        ├── ledger.py
        └── transaction.py
└── tests
    ├── test_crypto
    │   ├── __init__.py
    │   └── test_hashes.py
```

```

    └── test_keys.py
    └── test_sign_verify.py

└── test_settling
    ├── __init__.py
    ├── test_flow.py
    ├── test_graph.py
    └── test_Path.py

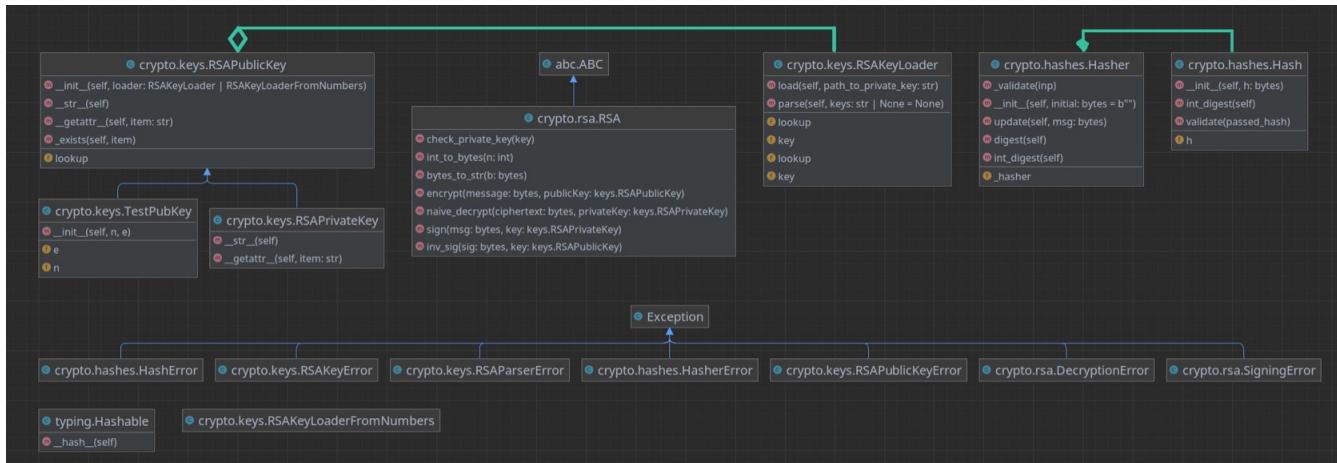
└── test_transactions
    ├── __init__.py
    ├── mock_db.csv
    ├── test_ledger.py
    └── test_transaction.py

```

The interactions of different classes are shown diagrammatically below. If I were to provide an entire class diagram for the system, it would be unreadable. Hence, I have broken diagrams down by module, referencing objects from other modules where appropriate.

Class Diagrams by Module

Cryptography (`crypto`) Module



Key for this, and all following class diagrams: red *m* indicates a method, yellow *f* indicates a field. Method signatures are included, with type hints of parameters where relevant to aid in highlighting relationships. Protected variables are denoted through name - they start with an underscore

Above is a class diagram for the `crypto` module.

The primary objective of this module is to handle all the security needed by the application. This mainly involves a consistent way to ensure the validity of transactions, as well as their origin. It also will take care of hashing the passwords of users and groups so that they are not stored in plaintext. This is a more minor role, however.

The `RSAPublicKey` and `RSAPrivateKey` objects are lightweight. Their only field is a dictionary, which stores parts of the key, and an identifier. Since an RSA key needs three components to work (when implemented, as it is here, with modular exponentiation

encryption/decryption), each component is stored separately in this dictionary. The `__getattr__` method will be overwritten from the `object` parent class so that it is possible to access parts of the key as one would access an attribute (i.e. for the modulus `RSAPublicKey.n`)

How RSA Works

RSA is an asymmetric encryption algorithm which works by taking the plaintext (encoded by an integer), raising it to a very large number, and then taking a modulus of a different very large number

$$c = p^e \mod n \quad (1)$$

where c is the ciphertext, p is the plaintext, e is the public exponent, and n is the modulus. How the these numbers are generated is beyond the scope of this project. To decrypt, a similar relationship is used

$$p = c^d \mod n \quad (2)$$

where d represents the private exponent.

In public key cryptography, everyone has access to the modulus and public exponent in a key, but it is very important that no one except the owner has access to the private exponent. For this reason, I will ensure that the private exponent is never sent across the network.

To create digital signatures, the process is similar. First, a hash of the message is generated. This is done by the `Hasher` object above. Then, equation (2) is used, with c representing the hash of the message in integer form, as opposed to the ciphertext. Similarly, p now represents the digital signature instead of the plaintext. The signature is then appended to the message.

To verify the digital signature, equation (1) is used. If the signature is valid, the output of this equation should be the hash of the message. Thus, one hashes the messages and compares it to the outcome of equation (1). If the two match, then the signature is valid. However, if they don't match, either the message has been tampered with or the signature has been tampered with.

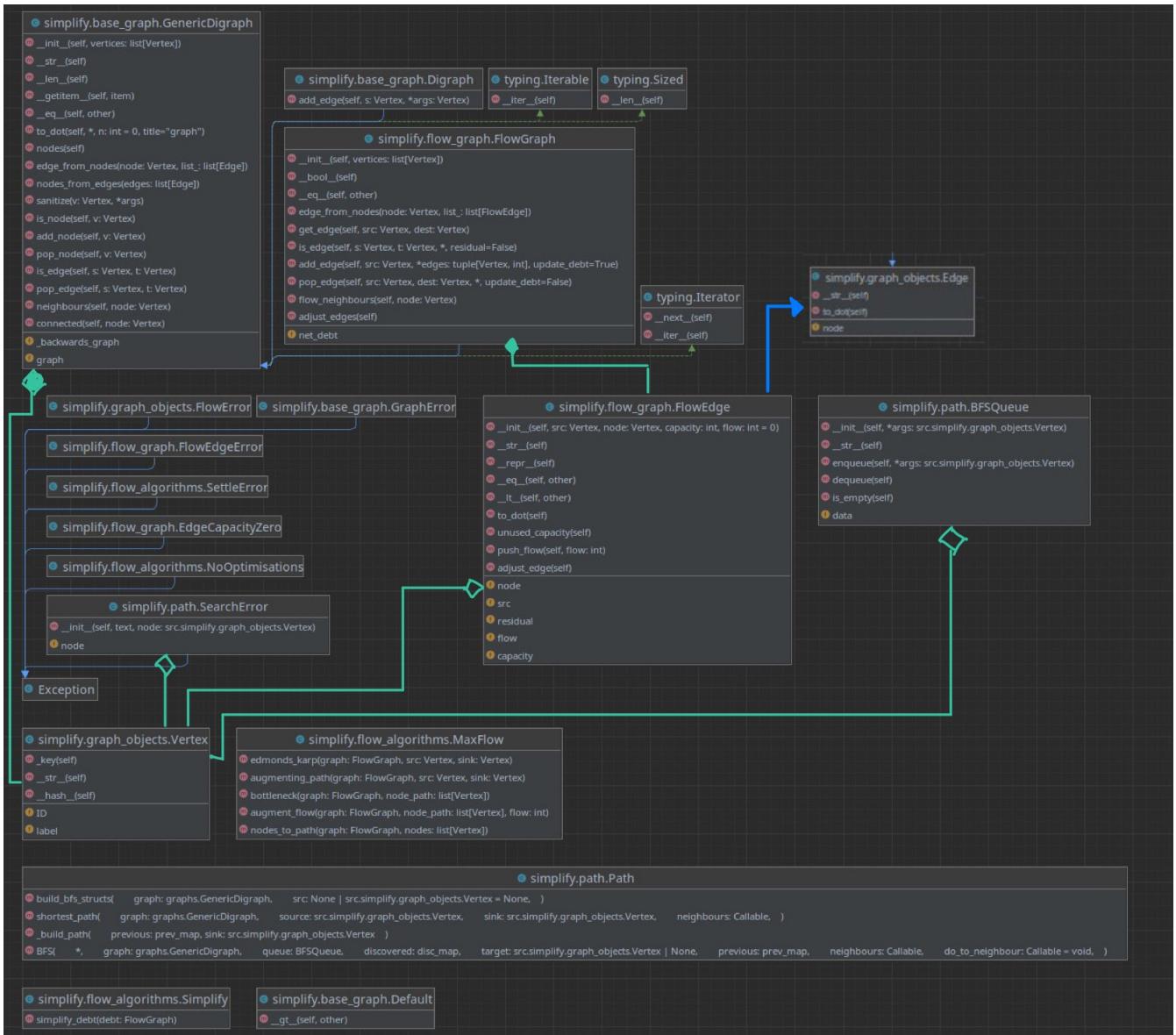
Thus, it is possible to ensure validity, and verify the origin, of messages.

I will implement this exactly in this way, using the Python builtin `pow()` to carry out the modular exponentiation, and the builtin `hashlib` library to generate hashes. I will, however, be writing my own interface to `hashlib` to add extra functionality, such as ensuring that all hashes that I generate are padded correctly so that the encryption / decryption works as expected.

I will also implement the loading of the RSA key in PEM format myself, using regexes to filter out only what I need, and package the resulting numbers into keys accordingly.

Debt Simplification (`simplify`) Module

Class Diagram



This is by far the most intricate module. It shows a complex object-oriented model (furthered in the `transactions` module). It also involves complex key data structures, as well as various complex algorithms - both well known and user defined.

The debt simplification module is dedicated to simplifying a graph using the algorithm specified in the analysis section. It does not have any dependencies on any other modules from this project.

The task of debt simplification is decomposed into four key areas: The flow graph data structure, graph search algorithms, graph flow algorithms, and the assembly of the main simplification interface.

The Flow Graph Data Structure

As I outlined in the project critical path, I wanted to have a basic unweighted digraph data structure, which I could perform breadth first searches (BFS) on before I started to consider

flow. To do this, I started with just a `GenericDigraph` class. The graph has only two fields: the `graph`, and the protected field `_backwards_graph`, to aid with the deletion of nodes.

The graph is effectively represented as an adjacency matrix. I use a dictionary, which maps a node to a list of `edges`. `edge` objects contain a `node` field, which represents the destination node, i.e. where the edge is pointing.

The base graph then has various bookkeeping methods such as checking if nodes are in the graph, checking if a node is associated a list of edges (and vice versa). You can also add and remove nodes and edges.

I designed a `neighbours` function to return the neighbours of a node, and a `connected()` function, seeing if a node has any connections in the graph.

I mentioned that the `_backwards_graph` was necessary when it comes to deleting nodes. This is because there must be a way of traversing the graph backwards to find which nodes are pointing to the node that you want to delete. If you do not protect against this, then you will end up with edges pointing nowhere.

The `_backwards_graph` will be managed every time an edge is added / deleted from the graph. When an edge(u, v) is added to the forwards graph, an edge(v, u) is added to the backwards graph. Then, when deleting a node, all that needs to be done is look at the connections of the given node in the backwards graph, and delete the edges from those nodes in the forwards graph.

This should provide everything necessary to implement breadth first search. I opted to do this recursively. The function signature did not fit on the class diagram, so is included here

```
def BFS(
    *, # star indicates keyword-only args
    graph: graphs.GenericDigraph,
    queue: BFSQueue,
    discovered: disc_map,
    target: src.simplify.graph_objects.Vertex | None,
    previous: prev_map,
    neighbours: Callable,
    do_to_neighbour: Callable = void,
) -> prev_map:
```

Note: `prev_map` and `disc_map` are not objects but custom type aliases. They are equivalent to

```
prev_map = dict[src.simplify.graph_objects.Vertex,
                 src.simplify.graph_objects.Vertex | None]
]

disc_map = dict[
    src.simplify.graph_objects.Vertex, src.simplify.graph_objects.Vertex | bool
]
```

To do the BFS, I first require a queue data structure. I will implement one myself instead of using a builtin queue. A BFS queue is interesting, as it should not allow for the same element to be enqueued twice. Thus, the data structure I will create will be an ordered set with only two operations: `enqueue()` and `dequeue()`.

I also need a data structure to keep track of the nodes that had been previously discovered, and where they had been discovered from. This will be important to be able to reconstruct a path through the graph. For this, I will use a dictionary of type `dict[Vertex, Vertex]`, where keys are vertices in the graph and values are where they were discovered from.

Since the BFS will end up being used in more than one way (searching through the standard graph, or searching for augmenting paths as part of the maxflow algorithm), it is important to specify to it how to look for neighbours. Since functions are first class objects in Python, this is done by passing in a function as and when it is needed.

Similarly, with `do_to_neighbour`, different algorithms that use the BFS will require different things to be done to the neighbours of a node. Hence, this is specified when the function is called, as opposed to when it is defined.

Here is a python mock-up of how I intend to implement the BFS

```
@staticmethod
def BFS(
    *,
    graph: graphs.GenericDigraph,
    queue: BFSQueue,
    discovered: disc_map,
    target: src.simplify.graph_objects.Vertex | None,
    previous: prev_map,
    neighbours: Callable,
    do_to_neighbour: Callable = void,
) -> prev_map:

    # will only happen if no path to node
    if queue.is_empty():
        return previous

    else:
        # discover next node in queue
        current = queue.dequeue()
        discovered[current] = True

        # check we haven't been fed a standalone node (i.e. no forward or
        # backwards links)
        if not graph.connected(current):
            if not queue:
                raise SearchError("Cannot traverse a non connected node",
current)

            # if discovered target node return prev
            if current == target:
```

```

        return previous

    else:
        # otherwise, continue on
            # enqueue neighbours, keep track of whose neighbours
        they are given not already discovered
            # do passed in function to neighbouring nodes
            for neighbour in neighbours(current):
                if not discovered[neighbour.node]:
                    previous[neighbour.node] = current
                    queue.enqueue(neighbour.node)

        do_to_neighbour(current, neighbour.node)

    # recursive call on new state
    return Path.BFS(
        graph=graph,
        queue=queue,
        discovered=discovered,
        target=target,
        previous=previous,
        neighbours=neighbours,
        do_to_neighbour=do_to_neighbour,
    )
)

```

After I implement the BFS, I will move onto the max-flow algorithm.

This requires an updating the graph data structure, and the edge data structure. To do this, I will inherit from the `BaseGraph` and `Edge` that I will have already designed.

The key changes to the edges (in a new class `FlowEdge`) will be:

- 1) Edges will need two new fields: `flow: int` and `capacity: int`, where capacity is a non-negative integer, and flow is an integer
- 2) Edges will need a new method: `unused_capacity()` which will return `capacity - flow`
- 3) Flow graphs contain a residual edge, as discussed in the Analysis phase. This will be accounted for with a field `residual: bool`. Residual edges will be treated as such.
- 4) In order to ease transaction integration later, I will also have edges explicitly track their `src` and their destination (`dest`). This will allow me to build a list of transactions just from a list of edges, but will be discussed further later on.
- 5) An `__eq__` function is needed, allowing differentiation between residual edges [5]

The key changes to the graph, in the new class `FlowGraph`, will be:

- 1) A backwards graph is no longer needed, instead it will be possible to utilise residual edges to traverse the structure the wrong way when deleting nodes
- 2) Adding edges now entails adding a residual edge counterpart, as discussed in the Analysis section. Thus, when edges are removed, their residual edge also needs to be removed. Hence, the `add_edge` and `pop_edge` functions need to be overwritten to work with `FlowEdge` objects.
- 3) Any pair of nodes should be restricted to just one forward edge. Thus, if there exists an edge from $A \rightarrow B$ of weight 5, and an edge from $B \rightarrow A$ of weight 10 is added, the graph

should result in one edge from B → A with weight 5.

4) A `flow_neighbour()` method needs to be introduced, as valid neighbours in the max-flow algorithm are any edges with unused capacity. This is different to a valid neighbour in the BFS, which is any forward-pointing non-residual edge.

5) A function is also needed to adjust the edges in the flow graph to become an edge with no flow, and only unused capacity remaining. This is added under the identifier `adjust_edges()`.

6) A way of verifying that the simplifying has resulted in a fair graph, where people owe and are owed the same (net) amount of money as they originally were. This is done with the `net_debt` field, which is a `dict[Vertex, int]`

In the analysis section, I detailed how the Edmonds-Karp max-flow algorithm works. I will implement it exactly as described in the analysis section. All the methods which are involved in finding the max-flow through a flow graph will be contained in the `flow_algorithms.MaxFlow` object. I will decompose the task of finding the `max_flow` into 5 functions. Their signatures are listed below

```
class MaxFlow:

    @staticmethod
        def edmonds_karp(graph: FlowGraph, src: Vertex, sink: Vertex) -> int:
    ...

    @staticmethod
        def augmenting_path(graph: FlowGraph, src: Vertex, sink: Vertex) ->
list[Vertex]: ...

    @staticmethod
        def bottleneck(graph: FlowGraph, node_path: list[Vertex]) -> int: ...

    @staticmethod
        def augment_flow(graph: FlowGraph, node_path: list[Vertex], flow: int)
-> None: ...

    @staticmethod
        def nodes_to_path(graph: FlowGraph, nodes: list[Vertex]) ->
list[FlowEdge]: ...
```

`augmenting_path` makes use of the recursive breadth first search that will have been implemented to find the shortest path through the graph from the source node to the sink node(in terms of number of edges).

`nodes_to_path` will be used to convert the list of nodes returned by the BFS function to a path.

`bottleneck` will return the bottleneck of a path through the graph, and `augment_flow` will augment the flow of a path, changing the `flow` field in each `FlowEdge`.

These are all combined in the `edmonds_karp` function to return an integer - the maximum flow from the given source node to the given sink node.

A python mock up of the implementation of the `edmonds_karp` function is given here

```
def edmonds_karp(graph: FlowGraph, src: Vertex, sink: Vertex) -> int:
    max_flow = 0

    while aug_path := MaxFlow.augmenting_path(graph, src, sink):
        bottleneck = MaxFlow.bottleneck(graph, aug_path)
        max_flow += bottleneck

        MaxFlow.augment_flow(graph, aug_path, bottleneck)

    return max_flow
```

With all the components having been designed, it is possible to integrate the process entirely. The `Simplify` class has one method: `simplify_debt(graph: FlowGraph)`. This combines all of what is above into my user-defined algorithm to simplify the graph as a whole. Again, this algorithm works exactly as laid out in the analysis section.

For every edge in the graph, a max-flow is run between the nodes at either end of the edge. This changes the state of the graph, as augmenting the flow through the graph will change the flow on edges / residual edges.

After the max-flow is run, an edge is added to the clean graph with a weight of the max-flow, and the `adjust_edges()` method is run on the 'initial' graph (initial in inverted commas because, of course, its state will have been changed by the algorithm. Calling it 'initial' just differentiates it from the clean graph that I'm building along the way).

This process should continue until there are no more edges in the 'initial' graph.

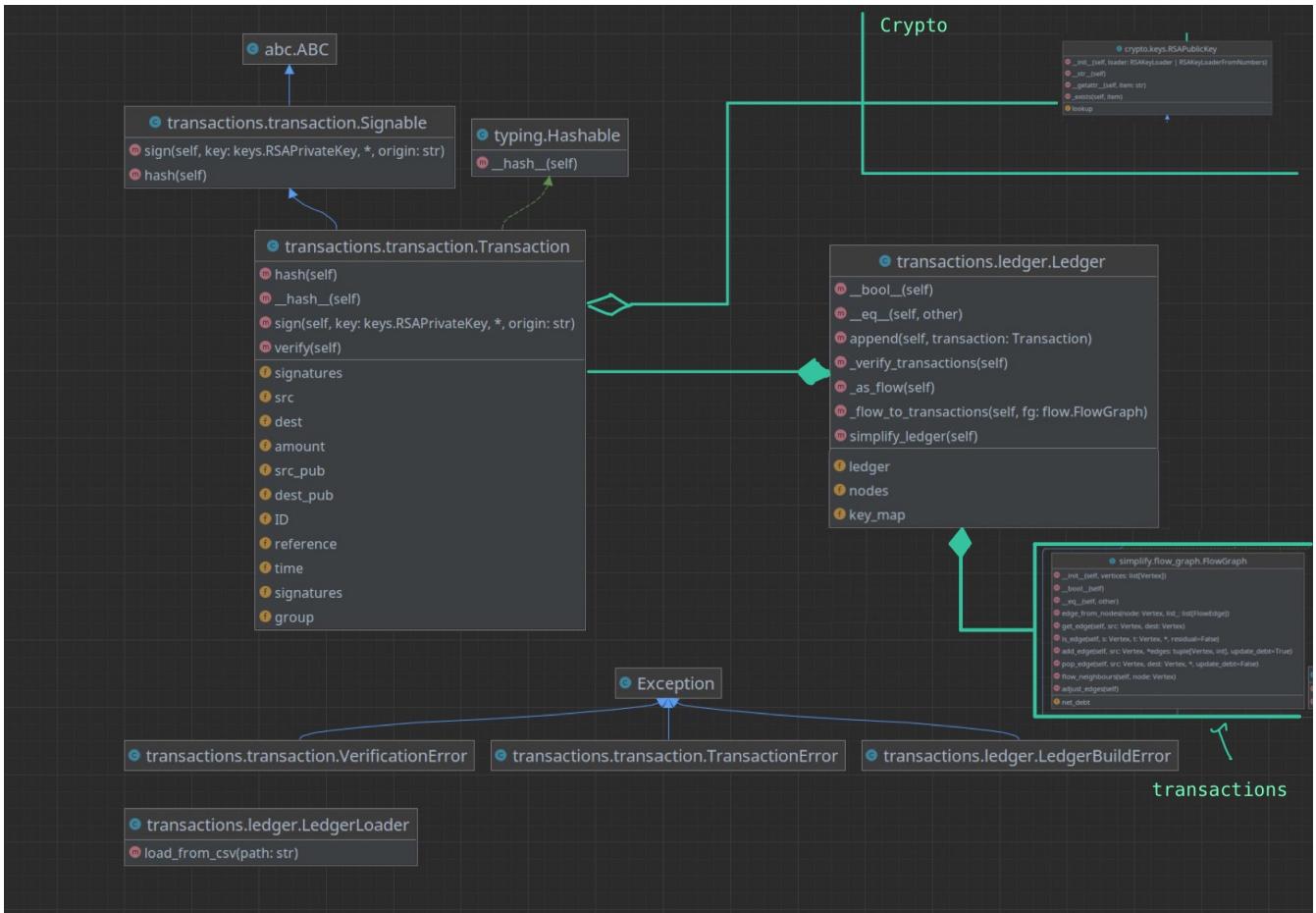
In pseudocode

```
for edge(u, v) in graph:
    if new := maxflow(u, v):
        clean.add_edge(u, (v, new))
        messy.adjust_edges()
```

Once the simplification has concluded, the net debts of the new graph should be compared with the (cached) net debts of how the graph was initially. These should always be identical, and the simplification process will raise an error if this is not the case, indicating that simplification should be aborted and retried.

This scenario is never expected to arise, but this failsafe should be implemented in favour of writing robust code.

Transaction integration (`transactions`) module



The purpose of the `transactions` module is to combine the `crypto` and `simplify` module - to allow transactions to be created, signed and verified.

A `Ledger` object will also be introduced to represent a group of transactions, and with the ability to simplify them, ensuring that they are all verified before doing so.

Since this module is effectively the assembly through aggregation and composition of the `crypto` and `simplify` modules, it does not have much in the way of complex data structures or algorithms.

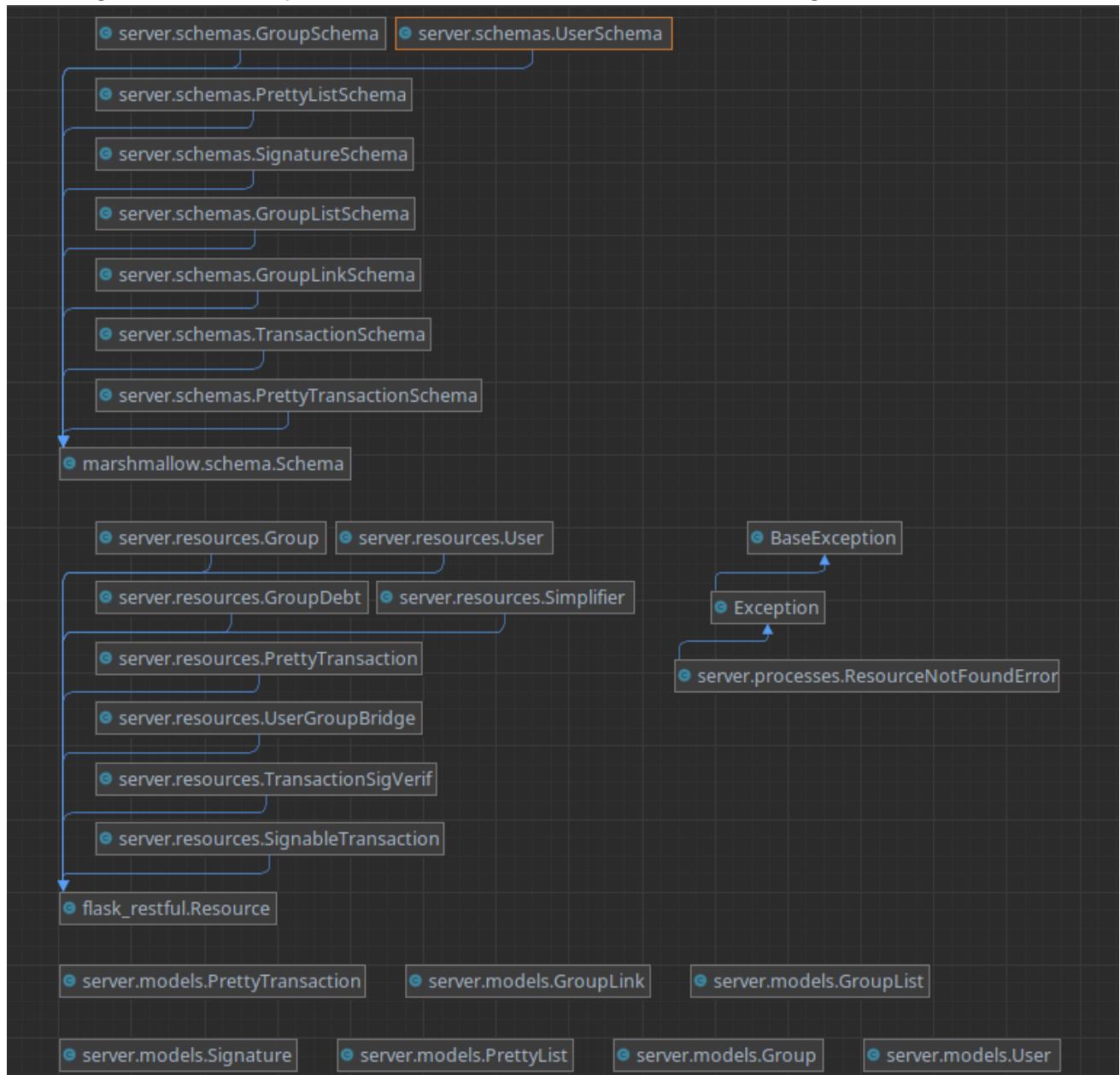
Signatures are stored as bytes inside the transaction object. Time is stored as a `datetime.datetime` object, and keys are stores as `crypto.keys.RSAPublicKey` objects (as per class diagram).

Ledger's `ledger` field is a list of `transaction` objects. Its `nodes` field is a list of all the people in the ledger, used to generate the flow graph that it creates to simplify transactions. `key_map` keeps track of which key belongs to which user ID.

The `Transaction` class inherits from the `Signable` class. This happens due to the order in which I intend to implement the project. The `crypto` module comes first, and I need to be able to test signing objects before I have transactions in place. This is also advantageous to me in case I decide to continue this project in the future and want to differentiate between different things that each need signing.

Server-side (**server**) module

On a high level, I anticipate the server module to have a class diagram as below



This is a high level overview of what resources, schemas, and models I will need to be able to transfer all the data that I need over my API.

Resources are the objects that sit behind endpoints - they each implement various HTTP methods. The design of some less obvious resources is discussed below.

API Endpoints and Resources

A lot of the work of the server is in serialising and deserializing objects / JSON. I will do this using the `marshmallow` library for Python. This requires that schema objects are set up with the same fields as the objects you want to serialise from / deserialize to.

This means that a lot of boilerplate code is needed, so I will not talk too much about that here as it is not particularly interesting, and does not prevent me from having a fully considered design of my problem.

I thought it to be more important to discuss the resources and endpoints that I would need to serve over my API. My resources are listed in the class diagrams above, and all serve an important purpose.

```
(Group, "/group/<int:id>", "/group")  
  
(PrettyTransaction, "/transaction", "/transaction/<string:email>")  
  
(User, "/user/<string:email>", "/user")  
  
(UserGroupBridge, "/group/<int:id>/<string:email>", "/group/<string:email>")  
  
(TransactionSigVerif, "/transaction/auth/<int:id>", "/transaction/auth/")  
  
(Simplifier, "/simplify/<int:gid>")  
  
(GroupDebt, "/group/debt/<int:id>")  
  
(SignableTransaction, "/transaction/settle/<int:t_id>",
"/transaction/signable/<int:id>")
```

This shows the **Resource** child classes as shown in the class diagram above with their endpoints.

A lot of the above resources implement GET and POST, which are self-explanatory by design in most cases (i.e. GET "user/<string:email>" will return user data for a given email"). I will discuss certain less obvious resources and endpoints.

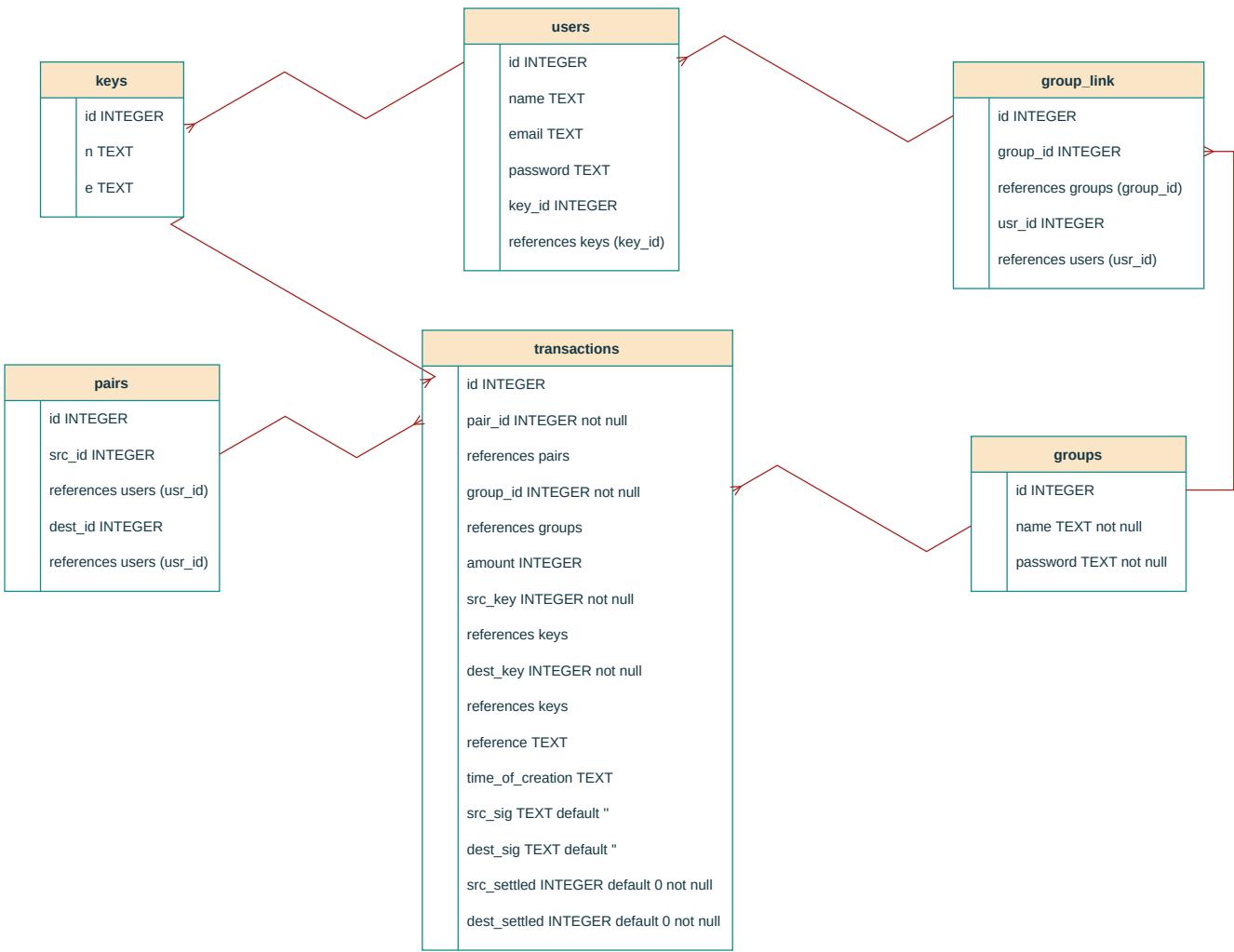
PrettyTransaction is a transaction object that is intended for being viewed on the front end by a user. It has people saved as emails as opposed to IDs, an association with a group, no keys involved, the time of creation, reference, and verification status. It also has the transaction ID. The POST method of pretty transaction is used to post new transactions to the database. This is because the details entered by the user about new transactions line up exactly with the pretty transaction schema. All processing such as adding public keys and user IDs is done by the server.

UserGroupBridge also warrants discussion. The resource implements POST and GET. POST will add a user to a group, and GET will get all groups associated with a given user.

TransactionSigVerif implements GET and PATCH. GET will verify a transaction, returning copy of verified transaction. PATCH will, upon receiving a signature, check that the signature is valid with data from the database (public key data, etc.), and if the signature is valid, the signature will be inserted into the database on the given transaction ID.

I intend to run the API using **flask** and **flask_restful**; two commonly used Python libraries for such purpose. During testing, I will run the server on **localhost**.

Database Access



Here is the same database's DDL

```

create table groups
(
    id      INTEGER
        primary key autoincrement,
    name    TEXT not null,
    password TEXT not null
);

create unique index groups_group_id_uindex
    on groups (id);

create table keys
(
    id INTEGER
        primary key autoincrement,
    n  TEXT,
    e  TEXT
);

create table sqlite_master
(
    type    text,
    name    text,
    tbl_name text,

```

```
    rootpage int,
    sql      text
);

create table sqlite_sequence
(
    name,
    seq
);

create table users
(
    id      INTEGER
        primary key autoincrement,
    name    TEXT,
    email   TEXT,
    password TEXT,
    key_id  INTEGER
        references keys (key_id)
);

create table group_link
(
    id      INTEGER
        primary key autoincrement,
    group_id INTEGER
        references groups (group_id),
    usr_id   INTEGER
        references users (usr_id)
);

create table pairs
(
    id      INTEGER
        primary key autoincrement,
    src_id  INTEGER
        references users (usr_id),
    dest_id INTEGER
        references users (usr_id)
);

create unique index uniq_pair
    on pairs (src_id, dest_id);

create table transactions
(
    id          INTEGER
        primary key autoincrement,
    pair_id     INTEGER not null
        references pairs,
    group_id    INTEGER not null
        references groups,
    amount      INTEGER,
    src_key     INTEGER not null
        references keys,
```

```
    dest_key      INTEGER not null
        references keys,
    reference     TEXT,
    time_of_creation TEXT,
    src_sig       TEXT      default '',
    dest_sig      TEXT      default '',
    src_settled   INTEGER default 0 not null,
    dest_settled  INTEGER default 0 not null
);

```

(There is more on the structure of the database in the Evaluation section.)

The server module will handle all interactions with the sqlite3 database (entity relationship diagram below)

Since this is a fairly complex relational database system, I put some thought into the queries that I would use to select data. Below is an example of such a query.

```
SELECT transactions.id, group_id, amount, reference, time_of_creation,
u2.email, verified
FROM transactions
INNER JOIN pairs p on p.id = transactions.pair_id
INNER JOIN users u on u.id = p.src_id
INNER JOIN users u2 on u2.id = p.dest_id
WHERE transactions.src_settled = 0
OR transaction.dest_settled = 0
AND u.email = ?;
```

Here is an example query intended to retrieve rows of data that can be used to build a `models.PrettyTransaction` object. Data is fetched where the user's email provided is the source of the transaction - i.e. it will retrieve a user's outgoing transactions.

This statement is not group specific, but will return all transactions that have not been settled associated with a user's email.

An important thing to keep in mind when designing my SQL statements is ensuring data integrity. Thus, I will take care to protect against adding duplicate records, and ensure that data is consistent. For instance, a transaction should not be able to be signed by a user who is not part of that transaction

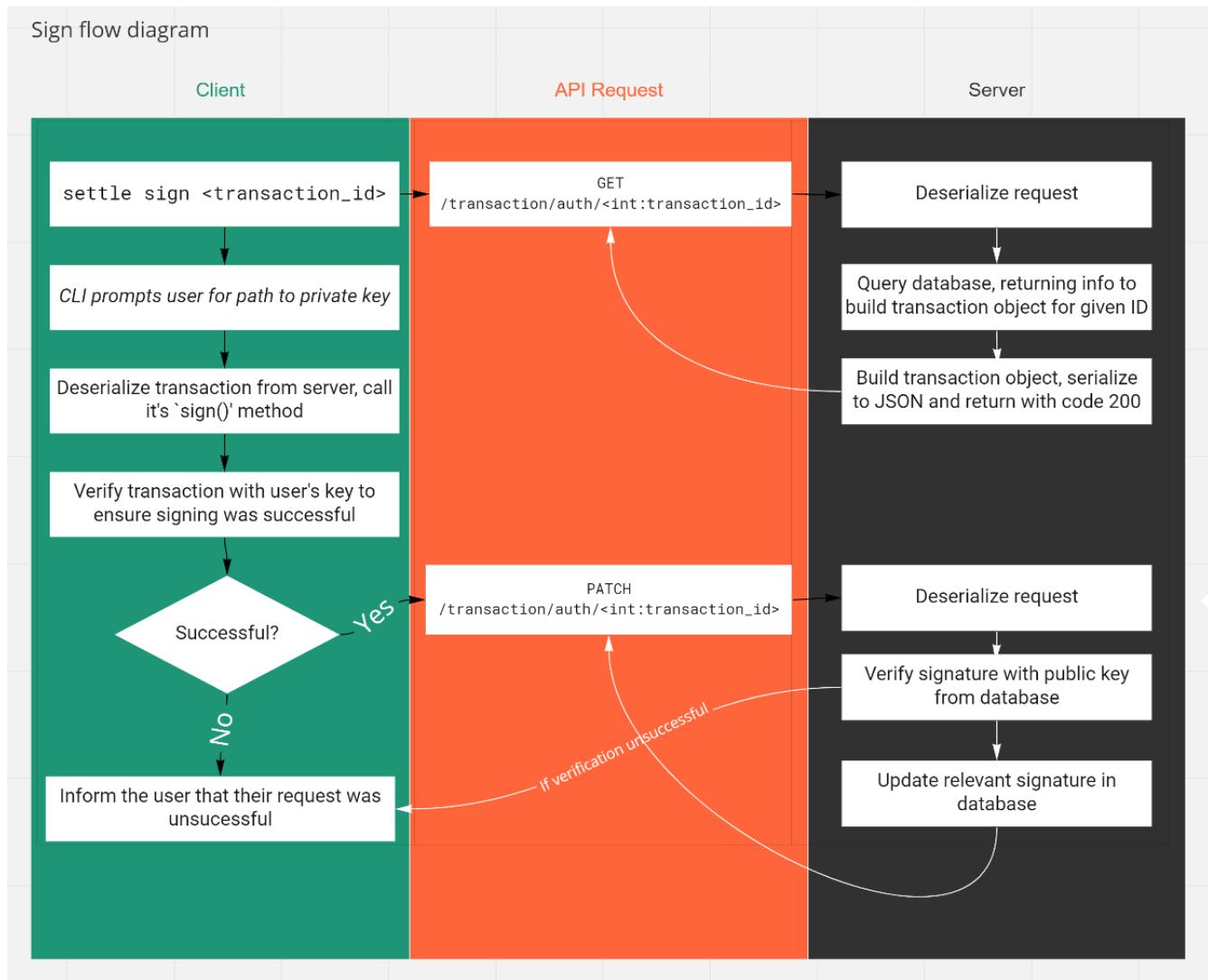
Server Logic

The server is, of course, more than just an API and database access - it will carry out the vast majority of the data processing. In that sense, my client server models is effectively thick server, thin client.

As with the endpoints, a lot of the logic is minimal and self-explanatory by design. Thus, here I will discuss two of the more interesting processes that the server can be asked to do.

Signing a transaction

Below is a swim lane diagram to aid my explanation of how a transaction is signed



This diagram shows the various processes that should run as a result of the user asking to sign a transaction.

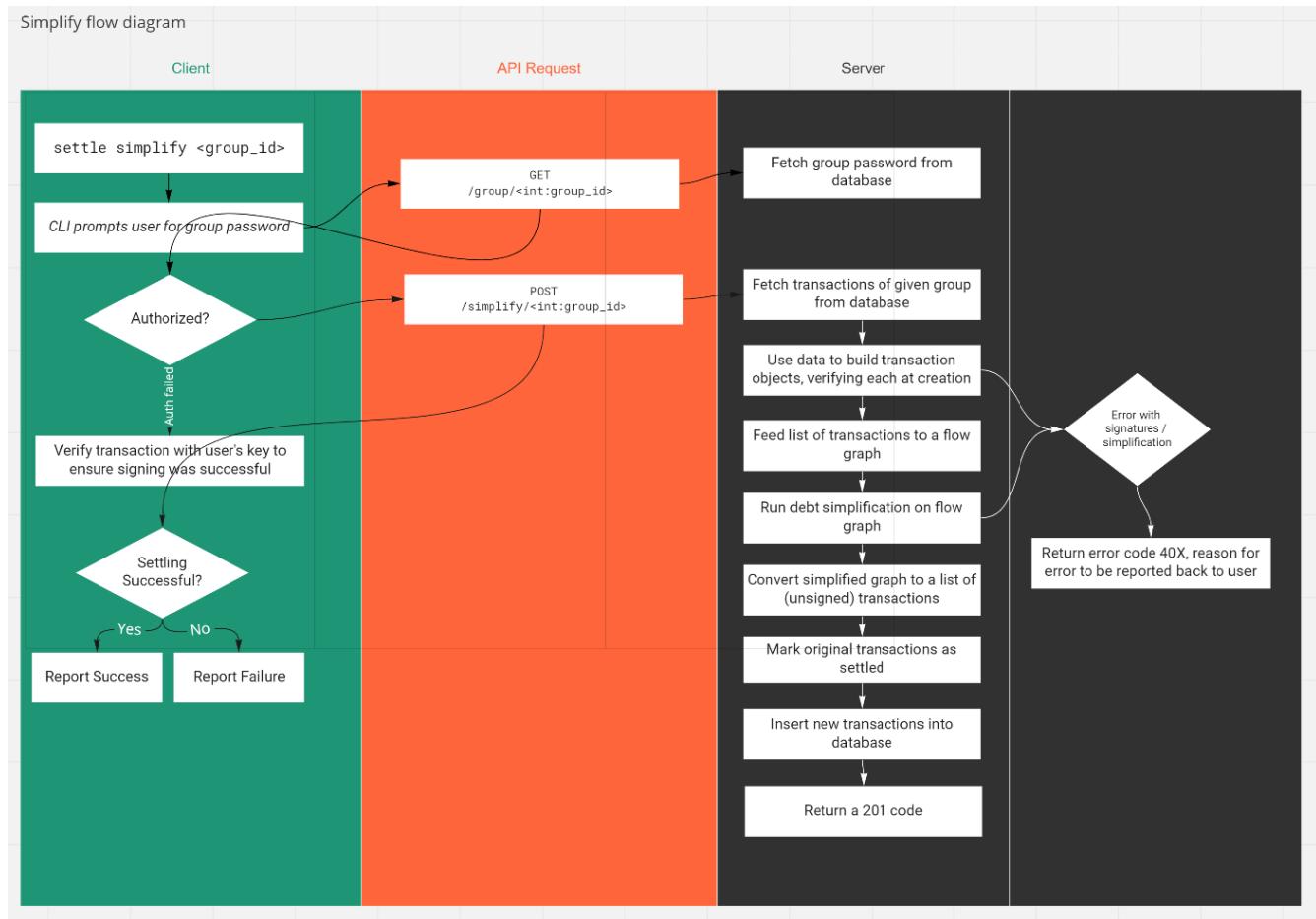
All client-server communication is done with JavaScript Object Notation (JSON) for this project.

For clarity, the diagram omits an argument to the `sign` command. `sign` also requires the user's email so that data can be kept in order in the database (more on this when the database design is discussed).

Here, 4 of the 5 modules are used. The `client` and `server` modules are clearly shown. The `transaction` module is used when constructing, signing and verifying transactions.

The `crypto` module is used in the `transaction` object, and provides the methods to be able to sign and verify the `transaction` object. It is also used to load keys on the client side (although this could be replaced with any PEM key loader).

Simplifying a group of transactions



This is in many ways the most important process in the project. Upon being asked to settle a group, every part of the project will be used.

In the implementation, all I will need to do is query the database with a query such as

```
SELECT transactions.id, src_id, dest_id, src_sig, dest_sig,
amount, reference, time_of_creation, group_id, k.e, k.n,
k2.e, k2.n
FROM transactions
JOIN keys k on k.id = transactions.src_key
JOIN keys k2 on k2.id = transactions.dest_key
JOIN pairs p on p.id = transactions.pair_id
WHERE transactions.group_id = ?;
```

to get all the data needed to build a `transaction.ledger.Ledger` of `transaction.transaction.Transaction` objects. Once those objects are built, I call `ledger.simplify_ledger()`. This will then invoke all the logic discussed in the `simplify` module section, as well as handle the verification of signatures, as discussed in the `crypto` module section. This will be wrapped in a `try: ... except: ...` clause, and any errors will be returned with a 40X error code and reason for failure.

Client-side (`client`) module

As aforementioned, the client is a thin client, meaning it does not have many responsibilities in the overarching structure of the program. Thus, this section will mainly be examples and mock-ups of the elements of Human-Computer Interaction.

To show this, I will provide screenshots of an output to STDOUT of how I would like certain outputs to look. Here I will provide mainly ancillary outputs. and how I would like certain prompts to appear upon a command being run

- Upon a user registering a new account

```
(venv) tcassar@ubuntu:~/projects/settle$ settle register
Full Name: Foo Bar
Email: foobar@example.com
Password:
Repeat for confirmation:
Error: The two entered values do not match.
Password:
Repeat for confirmation:
Path to RSA key: /home/tcassar/projects/settle/src/crypto/sample_keys/t_private-key.pem
Account created successfully

Name: Foo Bar
Email: foobar@example.com
Modulus: 0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca27
9500f4665bda3ca30d767baedb969d1ee532ad92c8d1388ec09d5553db32605785db7e3fc9aaeeb1e4235d8d5038bf046761
5a7e84442ff74ec0d952598174dfadab34908e99b2bc8746918752cf08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beaf
cf0779dcde5569da8a776376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb2754049facbd1efbaa6fc3b0a6e74fc
70fe984a85ac7e548c833da1fa40cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f
4c530b17a5046eeb21652cf55c605,
Public Exponent: 0x10001

(venv) tcassar@ubuntu:~/projects/settle$
```

The program should prompt the user with the details that they need to enter to create their account. Password entering should be hidden, as it is commonly in CLIs. Passwords should be confirmed through asking for confirmation, as above. If the passwords entered do not match, the program should ask the user re-enter their password. The program should report a failure if an invalid path to a key is given, as here

- Upon creating & joining a group

```
(venv) tcassar@ubuntu:~/projects/settle$ settle new-group
Name: Foo's Test Group
Password:
Repeat for confirmation:
"Created group ID=10 named Foo's Test Group"

You can join this group with `settle join`
(venv) tcassar@ubuntu:~/projects/settle$ settle join 10
Email: foobar@example.com
Your password:
Group Password:
Successfully joined group 10
```

- An example of how a failed attempt at an action should look is

```
(venv) tcassar@ubuntu:~/projects/settle$ settle new-transaction
Email of payee: foobar@example.com
Amount (in GBP): 12.99
Group: 3
Your email: cassar.thomas.e@gmail.com
Password:
Authorisation Error; aborting...
Password Incorrect
(venv) tcassar@ubuntu:~/projects/settle$
```

- Finally, viewing your existing transactions should look like this

```
(venv) tcassar@ubuntu:~/projects/settle$ settle show -t
Email: cassar.thomas.e@gmail.com

Your open transactions:

You owe keith@npl.com £12.99

Reference: scran
Agreed upon at 2022-03-19T16:53:37.720130
Verified

keith@npl.com owes you £12.99

Reference: scran2
Agreed upon at 2022-03-19T17:05:18.172690
Unverified
-----
You owe a total of £12.99
Your unverified totals => you are owed £12.99
(venv) tcassar@ubuntu:~/projects/settle$ █
```

This is not an exhaustive list, but is the general blueprint of how interaction should look. Every possible actions will end up looking like one of these above templates, be it something like simplifying or signing a transaction, which will just result in a confirmation, or seeing all the open transactions in the group, which will look the same as seeing all of your open transactions.

The full list of commands that I would like the user to be able to enter is below

```
Usage: settle [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  join            Joins a group given an ID
  new-group       Creates a new group given a name and email
  new-transaction Generates a new transaction
  register        Registers a new user
  show            Shows all of your open transactions / groups along...
  show-group      Shows the transactions in a group
  sign            Signs a transaction
  simplify        Simplifies debt of a group
  tick            Ticks off a transaction as settled up in the real world
  verify          Will verify a transaction if given a transaction ID or...
  whois           Shows the name, email and public key info given an email
```

This text should also be displayed when the `--help` flag is called after any of the commands.

Note: the `whois` command may seem odd - it allows you to obtain information about other people. What makes a system like this work is the fact that it is trust free. The system is to be designed so that everyone can see everyone's public keys and everyone can see everyone's transactions.

Like this, there is nowhere to hide - you will always be held accountable for your transactions.

Error Handling in the CLI

It is important that the user never experiences an unsightly looking crash message when an expected error in the program happens. For instance, if when registering for an account the user provides a path to a file that doesn't exist instead of their private key, they should not see the program crash. Instead, the program should prompt them that it could not complete the registration, because no file exists in that location. A mockup would look something like this.

```
(venv) tcassar@ubuntu:~/projects/settle$ settle register
Full Name: test
Email: test@test.com
Password:
Repeat for confirmation:
Path to RSA key: path/to/nowhere
Failed to create account - issue with given RSA key;
File not found at current path:
path/to/nowhere
(venv) tcassar@ubuntu:~/projects/settle$
```

-
1. based on a heuristic model ↵
 2. shortest here referring to fewest edges traversed (not accounting for edge weight) ↵
 3. ↵
 4. Due to the heuristic nature of the model there will likely be multiple valid settled graphs
↵
 5. the `__eq__` function in the base `Edge` class was generated by the `@dataclass` decorator.
However, this would fail to differentiate residual edges due to the way that `dataclasses` generates dunder methods ↵

File - /home/tcassar/projects/settle/src/_init__.py

```
1 # coding=utf-8
2 import logging
3
4 logging.basicConfig(filename='./log')
```



```
1 # coding=utf-8
2
3 import click
4
5 import src.client.client as client
6
7
8 @click.group()
9 def settle():
10     ...
11
12
13 # |-----|
14 # |  USER  |
15 # |-----|
16
17
18 @click.option("--pub_key", prompt="Path to RSA key"
19               , type=click.Path())
20 @click.option("--password", prompt=True, hide_input=True,
21               confirmation_prompt=True)
22 @click.option("--email", prompt=True)
23 @click.option("--name", prompt="Full Name")
24 @settle.command()
25 def register(
26     name,
27     email,
28     password,
29     pub_key,
30 ):
31     client.register(name, email, password, pub_key)
32
33 @click.argument("email")
34 @settle.command()
35 def whois(email):
36     client.whois(email)
37
38 @click.option("-g", "--groups", flag_value="groups",
39               default=False)
```

```

39 @click.option("-t", "--transactions", flag_value="transactions", default=False)
40 @click.option("--email", prompt=True)
41 @settle.command()
42 def show(transactions, groups, email):
43     """Shows all of your open transactions / groups along with IDs"""
44     client.show(transactions, groups, email)
45
46
47 @click.option("--password", prompt=True, hide_input=True)
48 @click.option("--email", prompt=True)
49 @click.argument("key_path")
50 @click.argument("transaction_id")
51 @settle.command()
52 def sign(transaction_id, key_path, email, password):
53     """Signs a transaction"""
54     client.sign(transaction_id, key_path, email, password)
55
56
57 @click.option("-g", "--group", default=0)
58 @click.option("-t", "--transaction", default=0)
59 @settle.command()
60 def verify(group, transaction):
61     """Will verify a transaction if given a transaction ID or an entire group if given a group ID"""
62     client.verify(group, transaction)
63
64
65 @click.option("--password", prompt=True, hide_input=True, confirmation_prompt=True)
66 @click.option("--name", prompt=True)
67 @settle.command(name="new-group")
68 def new_group(name, password):
69     client.new_group(name, password)
70
71
72 @click.option(

```

```

73     "--group_password",
74     prompt="Group Password",
75     hide_input=True,
76 )
77 @click.option(
78     "--password",
79     prompt="Your password",
80     hide_input=True,
81 )
82 @click.option("--email", prompt=True)
83 @click.argument("group_id")
84 @settle.command()
85 def join(email, password, group_id, group_password):
86     """Joins a group given an ID"""
87     client.join(email, password, group_id,
88     group_password)
89
90 @click.option("--password", prompt="Group Password"
91 , hide_input=True)
91 @click.argument("group_id")
92 @settle.command()
93 def simplify(group_id, password):
94     """Simplifies debt of a group"""
95     client.simplify(group_id, password)
96
97
98 @click.option("--password", prompt=True, hide_input
99 =True)
100 @click.option("--email", prompt="Your email")
100 @click.option("--group", "-g", prompt=True)
101 @click.option("--reference", prompt=True)
102 @click.option("--amount", prompt="Amount (in GBP)")
103 @click.option("--dest_email", prompt="Email of
104 payee")
104 @settle.command(name="new-transaction")
105 def new_transaction(email, password, dest_email,
106 amount, group, reference):
107     """Generates a new transaction"""
107     client.new_transaction(email, password,

```

```
107 dest_email, amount, group, reference)
108
109
110 @click.option("--email", prompt=True)
111 @click.option("--group_id", prompt="Group ID")
112 @settle.command(name="show-group")
113 def show_group(email, group_id):
114     client.group_debt(group_id, email)
115
116
117 @click.option("--password", prompt=True, hide_input
    =True)
118 @click.option('--email', prompt=True)
119 @click.argument('transaction')
120 @settle.command()
121 def tick(email, transaction, password):
122     """Ticks off a transaction as settled up in the
real world"""
123     if transaction is None:
124         click.secho('Please provide a transaction
with the -t flag (--help for help)', fg='red')
125         return
126
127     client.tick(email, password, transaction)
```

```
1 # coding=utf-8
2 import copy
3 import sys
4
5 import click
6 import json
7 import requests
8
9 import src.client.cli_helpers as helpers
10 import src.crypto.keys as keys
11 import src.server.models as models
12 import src.server.schemas as schemas
13 import src.transactions.transaction as trn
14 from src.client.cli_helpers import show_transactions
15
16 trap = helpers.trap
17
18 #####
19 # ANCILLARY #
20 #####
21
22
23 @trap
24 def register(
25     name,
26     email,
27     password,
28     pub_key,
29 ):
30     """Register to settle, using email, passwd, and
31     an RSA public key"""
32
33     # extract and store modulus and pub exp as bytes
34     # , ready for db
35     ldr = keys.RSAKeyLoader()
36     try:
37         ldr.load(pub_key)
38         ldr.parse()
39         pub_key: keys.RSAPublicKey = keys.
40             RSAPublicKey(ldr)
41     except keys.RSAParserError as rsa_err:
```

```

39         click.secho(
40             f"Failed to create account - issue with
41             given RSA key; \n{rsa_err}",
42             fg="red",
43             bold=True,
44         )
45         return
46
47     password: str = helpers.hash_password(password)
48
49     usr = models.User(
50         name,
51         email,
52         hex(pub_key.n),
53         hex(pub_key.e),
54         password,
55     )
56
57     # build json repr of object
58     schema = schemas.UserSchema()
59     usr_as_json = schema.dump(usr)
60
61     response = requests.post(helpers.url("user"),
62                               json=usr_as_json)
63     try:
64         helpers.validate_response(response)
65     except helpers.InvalidResponseError:
66         print(response)
67         click.secho(f"Failed to create account under
68             email {email}", fg="yellow")
69         return
70
71
72 @trap
73 def whois(email):
74     """gives your name, email, public key numbers"""
75     usr_response: requests.Response = requests.get(

```

```
75 helpers.url(f"user/{email}"))
76     helpers.validate_response(usr_response)
77
78     # build a user from received data
79
80     schema = schemas.UserSchema()
81     usr = schema.load(usr_response.json())
82     click.secho(f"\nFound user with email {email}:\n", fg="green")
83     click.secho(str(usr))
84
85
86 @trap
87 def show(transactions, groups, email):
88     """Shows all of your open transactions / groups
89     along with IDs"""
90
91     # note: flags are None or True for some
92     # godforsaken reason
93
94     # show both if no flags
95     if not transactions and not groups:
96         transactions = groups = True
97
98     # if groups:
99     #     click.secho("Groups that you are a member
100    # of:\n", fg="blue")
101
102    # receive list of groups JSON;
103    try:
104        groups_data = requests.get(helpers.url(
105            f"group/{email}"))
106    except helpers.ResourceNotFoundError as ire:
107        raise helpers.ResourceNotFoundError(
108            f"Problem fetching your group...\n{ire}")
109
110    group_objs: list[models.Group] = []
111    for group in groups_data.json()["groups"]:
112        group_objs.append(models.Group(**group))
```

```
109         group_objs.append(  
110             models.Group(group["id"], group["  
    name"], group["password"]))  
111     )  
112  
113     groups = models.GroupList(group_objs)  
114  
115     click.echo(str(groups))  
116     # type: ignore  
117  
118     if transactions:  
119  
120         # transaction schema  
121  
122         # receive list of transactions  
123         try:  
124             transactions_data = requests.get(  
    helpers.url(f"/transaction/{email}"))  
125             helpers.validate_response(  
    transactions_data)  
126         except helpers.ResourceNotFoundError as ire  
        :  
127             raise helpers.ResourceNotFoundError(  
128                 f"Problem with fetching your  
    transactions...\n{ire}"  
129             )  
130         show_transactions(transactions_data)  
131  
132  
133 @trap  
134 def join(email, password, group_id, group_password  
    ):  
135  
136     # 1: verify user  
137     helpers.auth_usr(email, password)  
138  
139     # 2: verify group  
140     helpers.auth_group(group_id, group_password)  
141  
142     # 3: post to groups  
143     group = requests.post(helpers.url(f"group/{"
```

```
143 group_id}/{email}))  
144     helpers.validate_response(group)  
145     click.secho(f"Successfully joined group {  
146         group_id}", fg="green")  
147  
148 @trap  
149 def new_group(name, password):  
150     schema = schemas.GroupSchema()  
151     # note: 0 is placeholder, will be overwritten  
152     # by db  
152     group = models.Group(0, name, helpers.  
153     hash_password(password))  
154     as_json = schema.dump(group)  
155     response = requests.post(helpers.url("group"),  
156     json=as_json)  
157     try:  
158         helpers.validate_response(response)  
159     except helpers.InvalidResponseError as ire:  
160         raise helpers.InvalidResponseError(f"Couldn't  
161         create new group...\n{ire}")  
162         click.secho(response.text, fg="green")  
163         click.secho("You can join this group with `  
164             settle join`")  
165  
166 #####  
167 # FUNCTIONAL #  
168 #####  
169  
170 #####  
171 # FUNCTIONAL #  
172 #####  
173  
174  
175 @trap  
176 def new_transaction(email, password, dest_email,  
    amount, group, reference):
```

```
177      # 1. verify src credentials
178      helpers.auth_usr(email, password)
179
180      # 2. get users as models.User objects
181      destination exists
182      src = helpers.get_user(email)
183      dest = helpers.get_user(dest_email)
184
185      # convert amount into pence
186      amount = float(amount)
187      amount *= 100
188      if type(amount) == float:
189          amount // 1
190          amount = int(amount)
191
192      # build key objects
193      src_key_ldr = keys.RSAKeyLoaderFromNumbers()
194      dest_key_ldr = keys.RSAKeyLoaderFromNumbers()
195
196      src_key_ldr.load(n=int(src.modulus, 16), e=int(
197          src.pub_exp, 16))
198      dest_key_ldr.load(n=int(dest.modulus, 16), e=
199          int(dest.pub_exp, 16))
200
201      # build transaction
202
203      transaction = trn.Transaction(
204          src=src.id,
205          dest=dest.id,
206          amount=amount,
207          src_pub=src_key,
208          dest_pub=dest_key,
209          reference=reference,
210          group=group,
211      )
212
213      # build schema
214      trn_schema = schemas.TransactionSchema()
```

File - /home/tcassar/projects/settle/src/client/client.py

```
215
216     # post to server
217     response = requests.post(
218         helpers.url("transaction"), json=trn_schema
219         .dump(transaction)
220     )
221
222     try:
223         helpers.validate_response(response)
224     except helpers.InvalidResponseError as ire:
225         raise helpers.InvalidResponseError(f"Failed
226         to add transaction\n{ire}")
227
228         click.secho(f"Transaction generated with ID={
229             response.json()}", fg="green")
230
231 @trap
232 def simplify(group_id, password):
233     """Will settle the group; can be done by anyone
234     at anytime;
235     everyone signs newly generated transactions if
236     new transactions are generated"""
237
238     helpers.auth_group(group_id, password)
239
240     response = requests.post(helpers.url(f"/
241         simplify/{group_id}"))
242
243     try:
244         helpers.validate_response(response)
245     except helpers.ResourceNotFoundError as ire:
246         raise helpers.ResourceNotFoundError(f"
247             Problem settling group... \n{ire}")
248
249     except helpers.NoChanges as nc:
250         raise nc
251
252     click.echo(f"Signed transaction with ID={response.
253         json()}"')
```

```

248
249 @trap
250 def sign(transaction_id, key_path, email, password):
251     """Signs a transaction given an ID and a path
252     to key"""
253
254     # auth user
255     helpers.auth_usr(email, password)
256
257     # load private key
258     ldr = keys.RSAKeyLoader()
259     try:
260         ldr.load(key_path)
261         ldr.parse()
262         key: keys.RSAPrivateKey = keys.
263             RSAPrivateKey(ldr)
264     except keys.RSAParserError as rsa_err:
265         click.secho(
266             f"Failed to sign transaction - issue
267             with given RSA key; \n{rsa_err}",
268             fg="red",
269             bold=True,
270         )
271     return
272
273     # pull signable transaction
274     response = requests.get(helpers.url(f"
275         transaction/signable/{transaction_id}"))
276
277     try:
278         helpers.validate_response(response)
279     except helpers.InvalidResponseError as ire:
280         raise helpers.InvalidResponseError(f"Failed
281         to fetch group data\n{ire}")
282
283     transaction: trn.Transaction = schemas.
284     TransactionSchema().make_transaction(
285         response.json()
286     )

```

```

282
283     # determine origin
284     usr_response: requests.Response = requests.get(
285         helpers.url(f"user/{email}"))
286     helpers.validate_response(usr_response)
287
288     # build a user from received data
289     usr = schemas.UserSchema().make_user(
290         usr_response.json())
291
292     # validate key in provided path against key
293     # from db
294     key_as_str = f"n={key.n},\nne={key.e}"
295
296     if usr.id == transaction.src:
297         origin = "src"
298         if transaction.src_pub.strip() != key_as_str.strip():
299             raise helpers.AuthError(
300                 "Private key provided does not
301                 match the listing in the db"
302             )
303     else:
304         click.secho("\tKey validated against
305                 server")
306
307     elif usr.id == transaction.dest:
308         origin = "dest"
309         if transaction.dest_pub.strip() != key_as_str.strip():
310             raise helpers.AuthError(
311                 "Private key provided does not
312                 match the listing in the db"
313             )
314     else:
315         click.secho("\tKey validated against
316                 server", fg="green")
317
318     else:
319         raise helpers.ResourceNotFoundError(
320             "Email provided doesn't match any of
321             users in the database")

```

```

313 the users in the transaction"
314     )
315
316     try:
317         # convert keys of sigs to ints to enable
318         # overwrite protection
319         as_strs = copy.deepcopy(transaction.
320             signatures)
321         transaction.signatures = {}
322         for s_key, s_val in as_strs.items():
323             if type(s_key) is str:
324                 transaction.signatures[int(s_key
325 )] = s_val
326
327             transaction.sign(key, origin=origin)
328             click.secho("\tsuccessfully signed
329             transaction", fg="green")
330             except trn.TransactionError as te:
331                 raise helpers.AuthError(f"Failed to sign
332             transaction: {transaction.ID}\n{te}")
333
334             # convert signature to hex
335             # todo: add sig_as_hex to transaction obj
336             sig_hex = hex(int.from_bytes(transaction.
337                 signatures[usr.id], sys.byteorder))
338
339             # patch signature
340             schema = schemas.SignatureSchema()
341             signature = models.Signature(transaction_id,
342                 sig_hex, origin)
343             rep = requests.patch(helpers.url("transaction/
344                 auth/"), json=schema.dump(signature))
345
346             if rep.status_code == 201:
347                 click.secho("\tSuccessfully appended
348                 signature in database!", fg="green")
349
350
351
352     @trap
353     def verify(groups, transactions: int):
354         """Will show you the transactions of a group,

```

```
344 or verify a transaction passed in by ID"""
345
346     if not groups and not transactions:
347         click.secho("Please provide a groups ID or
348             transaction ID", fg="yellow")
349
350     if transactions:
351         response = requests.get(helpers.url(f"
352             transaction/auth/{transactions}"))
353
354         try:
355             helpers.validate_response(response)
356         except helpers.InvalidResponseError as ire:
357             raise helpers.InvalidResponseError(f"
358                 Error in getting transaction, {ire}")
359
360             schema = schemas.PrettyTransactionSchema()
361             schema.make_pretty_transaction(response.
362                 json()).secho()
363
364             if groups:
365                 try:
366                     transactions_data = requests.get(
367                         helpers.url(f"group/debt/{groups}"))
368                     helpers.validate_response(
369                         transactions_data)
370
371                     except helpers.ResourceNotFoundError as ire
372 :
373
374                     raise helpers.ResourceNotFoundError(
375                         f"Problem with fetching your
376                         transactions...\n{ire}")
377
378
379             pretty_schema = schemas.PrettyListSchema()
380             # print(transactions_data.json())
381             pretty_list: models.PrettyList =
382                 pretty_schema.make_pretty_list(
383                     transactions_data.json()
384                 )
385             trn_schema = schemas.
```

```
375 PrettyTransactionSchema()
376
377         for trn in pretty_list.src_list +
378             pretty_list.dest_list:
379             trn_schema.make_pretty_transaction(trn)
380             .secho()
381
382 @trap
383 def tick(email: str, password: str, t_id: int):
384     # auth user
385     helpers.auth_usr(email, password)
386
387     rep = requests.patch(helpers.url(f"transaction/
388         settle/{t_id}"), json=json.dumps({'email': email},
389         indent=4))
390     helpers.validate_response(rep)
391     click.secho(f'Transaction {t_id} marked as
392         settled!', fg='green')
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2296
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2387
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2488
2489
2490
2491
2492
2493
2494
2495
2496
2496
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
```

File - /home/tcassar/projects/settle/src/client/__init__.py

```
1 # coding=utf-8
2 import logging
3
4 logging.basicConfig(level=logging.CRITICAL)
5
```



```
1 # coding=utf-8
2 import click
3 import requests
4
5 from src.crypto import hashes as hasher
6 from src.server import models as models, schemas as
schemas
7
8 SERVER = "http://127.0.0.1:5000/"
9
10
11 class AuthError(Exception):
12     ...
13
14
15 class ResourceNotFoundError(Exception):
16     """Resource could not be found on the server (404)"""
17
18
19 class InvalidResponseException(Exception):
20     """Requested action was understood but not allowed (403 / 409)"""
21
22
23 class ServerError(Exception):
24     ...
25
26
27 class NoChanges(Exception):
28     ...
29
30
31 def hash_password(password) -> str:
32     return str(hasher.Hasher(password.encode(
encoding="utf8")).digest().h)
33
34
35 def url(query: str) -> str:
36     return SERVER + query
37
```

```

38
39 def validate_response(response: requests.Response
) -> None:
40     """Raises InvalidResponseError if response is
invalid"""
41     e = response.text
42     if response.status_code == 404:
43         raise ResourceNotFoundError(
44             f"ERROR: Could not find requested
resource on server, {e}"
45         )
46     elif response.status_code == 409:
47         raise ResourceNotFoundError(
48             f"The resource that you are trying to
create already exists, {e}"
49         )
50     elif response.status_code == 403:
51         raise ResourceNotFoundError(f"This action
was not allowed by the server, {e}")
52     elif response.status_code // 100 == 5:
53         raise ServerError(f"Error {response.
status_code}: {response.json()['message']}")
54     elif response.status_code == 202:
55         raise NoChanges(response.text)
56
57
58 def _auth(resource: str, password: str):
59     usr_response: requests.Response = requests.get(
url(resource))
60     try:
61         validate_response(usr_response)
62     except ResourceNotFoundError:
63         raise ResourceNotFoundError(
64             f"No {resource.split('/')[0]} with
identifier {resource.split('/')[1]} found"
65         )
66
67     # build a user from received data
68
69     rep = usr_response.json()
70

```

```

71     if rep["password"] != hash_password(password):
72         raise AuthError("Password Incorrect")
73
74
75 def auth_usr(email: str, password: str):
76     """Authorises user, raises AuthError if fails
77     """
78
79
80 def auth_group(group_id: int, password: str):
81     return _auth(f"group/{group_id}", password)
82
83
84 def get_user(email: str) -> models.User:
85     usr_rep = requests.get(url(f"user/{email}"))
86
87     try:
88         validate_response(usr_rep)
89     except ResourceNotFoundError:
90         raise ResourceNotFoundError(f"No user
associated with email {email}")
91
92     usr = usr_rep.json()
93
94     return models.User(
95         usr["name"], usr["email"], usr["modulus"],
96         usr["pub_exp"], "", usr["id"]
97     )
98
99 def trap(func) -> object:
100     """
101     Decorator to handle errors on these functions
102     """
103
104     def inner(*args, **kwargs):
105         try:
106             func(*args, **kwargs)
107
108         except AuthError as ae:

```

```

109         click.secho("Authorisation Error;  

110         aborting...", fg="red")  

111         click.secho(ae, fg="red")  

112     except ResourceNotFoundError as nre:  

113         click.secho(  

114             f"{nre}",  

115             fg="yellow",  

116         )  

117  

118     except ServerError as se:  

119         click.secho(se, fg="red")  

120  

121     except NoChanges as nc:  

122         click.secho(f"No changes were made\n{nc  

123         }", fg="yellow")  

124     except requests.exceptions.ConnectionError:  

125         click.secho('ERROR: Could not connect  

126         to server', fg='red')  

127     return inner  

128  

129  

130 def show_transactions(transactions_data: requests.  

131     Response):  

132     try:  

133         unverified_running = 0.00  

134         verified_running = 0.00  

135         for pretty in transactions_data.json()["  

136             src_list"]:  

137             click.secho(  

138                 f'\nYou owe {pretty["other"]} £{  

139                     round(pretty["amount"] / 100, 2):.2f}',  

140                     fg="yellow",  

141             )  

142             click.secho(  

143                 f'\nReference: {pretty["time"]}'  

144                 + f'\nAgreed upon at {pretty[""

```

```

143     reference"]}'  
144             + f'ID: {pretty["id"]}'  
145         )  
146  
147         if pretty["verified"] == 1:  
148             click.secho("Verified", fg="green")  
149             verified_running += pretty["amount"]  
150         else:  
151             click.secho("Unverified", fg="red")  
152             unverified_running += pretty["  
153                 amount"]  
154         for pretty in transactions_data.json()["  
155             dest_list"]:  
156             click.secho(  
157                 f'\n{pretty["other"]} owes you £{  
158                     round(pretty["amount"] / 100, 2):.2f}',  
159                     fg="yellow",  
160             )  
161             click.secho(  
162                 f'\nReference: {pretty["time"]}'  
163                 + f'\nAgreed upon at {pretty["  
164                     reference"]}'  
165                     + f'ID: {pretty["id"]}'  
166             )  
167  
168         if pretty["verified"] == 1:  
169             click.secho("Verified", fg="green")  
170             verified_running -= pretty["amount"]  
171         else:  
172             click.secho("Unverified", fg="red")  
173             unverified_running -= pretty["  
174                 amount"]  
175         click.echo("-----\n")  
176         unverified_running = round(  
177             unverified_running / 100, 2)

```

```

176         verified_running = round(verified_running
177             / 100, 2)
178
179         if verified_running > 0:
180             click.secho(f"You owe a total of £{verified_running:.2f}", fg="red")
181         elif verified_running < 0:
182             click.secho(
183                 f"You are owed a total of £{verified_running * -1 :02}", fg="blue"
184             )
185         else:
186             click.secho(f"You owe and are owed nothing; all debts settled", fg="green")
187
188         if unverified_running > 0:
189             click.secho(
190                 f"Your unverified totals => you owe £{unverified_running:.2f}",
191                 fg="yellow",
192             )
193         elif unverified_running < 0:
194             click.secho(
195                 f"Your unverified totals => you are owed £{unverified_running * -1 :02}",
196                 fg="yellow",
197             )
198         else:
199             click.secho(f"Your unverified totals => all debts settled", fg="yellow")
200
201     except TypeError as te:
202         if transactions_data.json() is None:
203             click.secho("No open transactions", fg="green")
204
205
206     def show_group(transaction_data: requests.Response):
207         schema = schemas.PrettyListSchema()

```

```
208     pretty_list: models.PrettyList = schema.  
209         make_pretty_list(transaction_data.json())  
210         print(pretty_list)
```



```
1 # coding=utf-8
2 """
3 RSA Sign/Verify classes
4 """
5
6 from __future__ import annotations
7
8 import sys
9 from abc import ABC
10 from typing import TYPE_CHECKING
11
12 from src.crypto import keys
13
14 if TYPE_CHECKING:
15     pass
16
17
18 class DecryptionError(Exception):
19     """Failed to decrypt"""
20
21
22 class SigningError(Exception):
23     ...
24
25
26 class RSA(ABC):
27     """RSA methods; ABC so is never instantiated;
28     just used for namespacing"""
29
30     @staticmethod
31     def check_private_key(key) -> None:
32         if type(key) == keys.RSAPublicKey:
33             raise DecryptionError("Cannot decrypt
34             with a public key")
35
36     @staticmethod
37     def int_to_bytes(n: int) -> bytes:
38         return int.to_bytes(n, length=n.bit_length
39             (), byteorder=sys.byteorder).rstrip(
40                 b"\x00"
41             )
```

```
39
40     @staticmethod
41     def bytes_to_str(b: bytes) -> str:
42         return b.decode("utf8").replace("\x00", "")
43
44     @staticmethod
45     def encrypt(message: bytes, publicKey: keys.
46                 RSAPublicKey) -> bytes:
47         message = int.from_bytes(message, sys.
48                                   byteorder)
49         cipher = pow(message, publicKey.e, publicKey
50                     .n)
51
52         return RSA.int_to_bytes(cipher)
53
54     @staticmethod
55     def naive_decrypt(ciphertext: bytes, privateKey
56                        : keys.RSAPrivateKey) -> bytes:
57         # TODO: CRT decryption
58
59         RSA.check_private_key(privateKey)
60
61         ciphertext = int.from_bytes(ciphertext, sys.
62                                     byteorder)
63         plaintext = pow(ciphertext, privateKey.d,
64                         privateKey.n)
65
66         return RSA.int_to_bytes(plaintext)
67
68     @staticmethod
69     def sign(msg: bytes, key: keys.RSAPrivateKey
70              ) -> bytes:
71         # check we have a private key
72
73         # extract integer if sign given bytes
74
75         RSA.check_private_key(key)
76
77         msg = int.from_bytes(msg, sys.byteorder)
78         cipher = pow(msg, key.d, key.n) # type:
79
80         ignore
```

```
72
73         return RSA.int_to_bytes(cipher)
74
75     @staticmethod
76     def inv_sig(sig: bytes | int, key: keys.
77                 RSAPublicKey) -> bytes:
77         """Will produce what was originally fed
78             into sign() using public key
78             used in verifying; if verified, should
78             generate hash of obj"""
79         if type(sig) is bytes:
80             sig: int = int.from_bytes(sig, sys.
81             byteorder)
81
82         de_sig = pow(sig, key.e, key.n)
83
84     return RSA.int_to_bytes(de_sig)
85
```



```
1 # coding=utf-8
2 """Interface to all RSA encrypt / decrypt functions
3 """
4
5 import os
6 import os.path
7 import re
8 import subprocess
9 from dataclasses import dataclass, field
10
11 class RSAKeyLoaderFromNumbers:
12     ...
13
14
15 class RSAKeyError(Exception):
16     """Problem with RSA Key"""
17
18
19 class RSAParserError(Exception):
20     """Error in parsing file containing key"""
21
22
23 class RSA PublicKeyError(Exception):
24     ...
25
26
27 @dataclass
28 class RSAKeyLoader:
29     """Will load a key from a private key file"""
30
31     # initialise what we need, don't pass in
32     # anything at instantiation
33     lookup: dict[str, int] = field(default_factory=
34         lambda: {})
35     key: None | str = None
36
37     def load(self, path_to_private_key: str) -> str:
38         """
39             Loads PEM private key from file in file path
40         ;
```

```

38      """
39
40      # will only attempt to load if file exists
41      # if openssl rsa doesn't like file report as
42      # incorrect format
43
44      try:
45          if not os.path.exists(
46              path_to_private_key):
47              raise RSAParserError(
48                  f"File not found at current path
49                  : \n{path_to_private_key}"
50                  )
51          key = str(
52              subprocess.check_output(
53                  f"openssl rsa -noout -text < {
54                  path_to_private_key}", shell=True
55                  )
56          )
57          except subprocess.CalledProcessError:
58              raise RSAParserError("File not in PEM
59          private key format")
60
61          # strip into long easily parseable str; =>
62          # remove spaces, newlines, colons
63          key = key.replace(" ", "").replace("\n", "")
64          .replace(":", "")
65
66          # converted from bytes so remove b' ' with
67          # slice
68          key = key[2:-1]
69
70          # split after title section, discard info
71          # pub exp is given in bin in text format;
72          # breaks splitting header with )
73          key = re.sub(r"\(0x[0|1]*\)", "", key) #
74          matches of the form (0x[0|1]*
75          head, key = key.split(")")
76
77          if head != "RSAPrivate-Key(2048bit,2primes":
78              raise RSAParserError("File not in

```

```

68     correct format")
69
70         self.key = key
71         return key
72
73     def parse(self, keys: str | None = None) ->
74         None:
75             """Given loaded SSL info, parses and
76             populates n, d, e, p, q"""
77
78             def hexstr_to_int(hexstr: str) -> int:
79                 # deals with exception pub exp which is
80                 # always a decimal
81                 return int(hexstr) if re.fullmatch("[0-
82                 9]+", hexstr) else int(hexstr, 16)
83
84                 # use local key if key not given
85                 if keys is None:
86                     keys: str = self.key
87                     if self.key is None:
88                         raise RSAParserError("Key has not
89                         been loaded")
90
91                 # set up delimiters; tells us to split when
92                 # parsing
93                 delimiters = ["n", "e", "d", "p", "q", " "
94                 "exp1", "exp2", "crt_coeff"]
95
96                 # split into a list along delimiters
97                 keys = re.sub(
98                     "modulus|publicExponent|privateExponent
99                     |prime1|prime2|exponent1|exponent2|coefficient",
100                     " ", "
101                     keys,
102                     )
103                 keys: map = map(hexstr_to_int, keys.split
104                     ())
105
106                 # combine to dict
107                 self.lookup = {label: key for label, key in
108                     zip(delimiters, keys)}

```

```

99
100
101 @dataclass
102 class RSAPublicKey:
103     def __init__(self, loader: RSAKeyLoader |
104      RSAKeyLoaderFromNumbers):
105         self.lookup = loader.lookup
106
107     def __str__(self):
108         return f"n={self.n},\n e={self.e}\n"
109
110     def __getattr__(self, item: str) -> int:
111         """Redefine getattr so that will only give
112         n and e"""
113         if item == "n" or item == "e":
114             return self.lookup[item]
115         else:
116             raise RSAPublicKeyError("Requested
117             attribute not part of the Public Key")
118
119     def _exists(self, item) -> bool:
120         """Returns an attribute if it exists else
121         raise an RSAKeyError"""
122         if self.lookup is not None:
123             try:
124                 return not not self.lookup[item]
125             except KeyError:
126                 raise RSAKeyError(
127                     "Requested attribute not found
128 ; have you parsed a key?")
129
130
131     def load(self, n: int, e: int, d: int = 0) ->
132         None:
133         self.lookup["n"] = n

```

```
133         self.lookup["e"] = e
134     if d:
135         self.lookup["d"] = d
136
137     def pub_key(self) -> RSAPublicKey:
138         return RSAPublicKey(self)
139
140     def priv_key(self):
141         return RSAPrivatekey(self)
142
143
144 class RSAPrivatekey(RSAPublicKey):
145     def __str__(self):
146         return f"n={self.n},\ne={self.e},\nd={self.
147 d}"
148     def __getattr__(self, item: str) -> int:
149         """
150             Accepted items:
151             n
152             e
153             d
154             p
155             q
156             exp1
157             exp2
158             crt_coeff
159         """
160         if self._exists(item):
161             return self.lookup[item]
162         else:
163             raise RSAkeyError
164
165
166 class TestPubkey(RSAPublicKey):
167     def __init__(self, n, e):
168         self.n = n
169         self.e = e
170
```



```
1 # coding=utf-8
2
3 """
4 Simple RSA implementation for signing / verifying
5 digital signatures.
6 Will have side channel vulnerabilities due to the
7 way that Python stores numbers
8 Hashing is done through hashlib library
9 """
10
11 import hashlib
12 import sys
13 from dataclasses import dataclass
14
15 class HashError(Exception):
16     """Hash provided is not in valid format"""
17
18 class HasherError(Exception):
19     ...
20
21
22 @dataclass
23 class Hash:
24     def __init__(self, h: bytes):
25         self.validate(h)
26         self.h: bytes = h
27
28     def int_digest(self) -> int:
29         # int representation of bytes, not actual
30         value
31         return int.from_bytes(self.h, byteorder=sys.
32         byteorder)
33
34     @staticmethod
35     def validate(passed_hash):
36         try:
37             assert len(passed_hash) == 32 # 256 / 8
38             assert type(passed_hash) == bytes
39         except AssertionError:
```

File - /home/tcassar/projects/settle/src/crypto/hashes.py

```
38             raise HashError("Hash was not in the
39             form of a hash")
40
41     class Hasher:
42         """Interface to hashing; for consistency always
43         use SHA256, """
44
45         @staticmethod
46         def _validate(inp):
47             """Raises HasherError if input isn't bytes
48             """
49
50             if type(inp) is not bytes:
51                 raise HasherError(
52                     f"Type passed into _hasher was not
53                     bytes; received {type(inp)}"
54                 )
55
56         def __init__(self, initial: bytes = b"") -> None
57         :
58             self._validate(initial)
59             self._hasher = hashlib.sha3_256(initial)
60
61         def update(self, msg: bytes) -> None:
62             """Updates _hasher with given message after
63             checking that message was in fact bytes"""
64             self._validate(msg)
65             self._hasher.update(msg)
66
67         def digest(self) -> Hash:
68             """Digests current hash; digests always
69             returns a Hash object"""
70             return Hash(self._hasher.digest())
71
72         def int_digest(self) -> int:
73             return Hash(self._hasher.digest()).int_digest()
```

```
1 # coding=utf-8  
2
```



```
1 -----BEGIN PUBLIC KEY-----
2 MIIBIjANBgkqhkiG9w0BAQEFAOCAQ8AMIIBCgKCAQEaqzzsGK6N
9oxQLtcc4zds
3 ygDF6Cruv5qlBHl00rSzQAk+0By+
LchMJ02mQKLSw3moxHhogTlJpKecTusBS/PD
4 /VNYz2gjBLe4Db4jPyT59Ah40bAHI1z5C/PqH82b51TX0+
4RqC7MprCdSeeNXQc
5 L5V+10b30GIoyKkw4lCVRrnCclWF5u0E1YnDFjCsdKTW+
LcwY0QuR7vq2PHpbeW0
6 m8dJz1Y2zuXTxc27ztqGPcriFrtm7DkV/3Li9g6WsT1AQ0000F+
SaiW66+u2zNc
7 9jST2gKzKSYMktcYPHnusblnRzlCpLpB8hPHEkc2ApdZRvoSc+Mq
+9kjUMyXEy/
8 qwIDAQAB
9 -----END PUBLIC KEY-----
```

10


```
1 -----BEGIN PUBLIC KEY-----
2 MIIBIjANBgkqhkiG9w0BAQEFAOCAQ8AMIIBCgKCAQEAssGOMnKA
 /0q+PtM3sCJR
3 hWOjNocYURePnuE82Ahch1wIAYTYNs0KPy0ft3HJiTGNviWC/
 G6iJBNTvcznuZz
4 0UwtkVe8Pq/iZuK976s/Yg+
 1SIrTqETTUbwNhWEL4kjCzI4VEmrjgdnKnZKHDxo
5 cSnwZUXDv55HNTpAi5CGNwFVQnaKxORDp3nhMY5vnSb0s2vpAEx
 KjKeJydhWLAh
6 fJP5qEDxD4q0aJQJyZbXsvT+tmm4QnuF+
 ptGwDmG3xAZA9Lbtk6xe0plUMzQOKnN
7 nm9CUBEGo0YAVyGWv9ymL/
 ucPIaCDdjW0kR0RuMqqqLeiROGjs3BAomkk548Wzdu
8 FwIDAQAB
9 -----END PUBLIC KEY-----
```

10


```
1 -----BEGIN PUBLIC KEY-----
2 MIIBIjANBgkqhkiG9w0BAQEFAOCAQ8AMIIBCgKCAQEAmmwTyC9d
846+33c1YUSk
3 cd+x9i/3j292+v00wUp1WWA/
ceJvVbtkyieVAPRmW9o8ow12e67blp0e5TKtksjR
4 OI7AnVVT2zJgV4Xbfj/Jqu6x5CNdjVA4vwRnYVp+
hEqv907A2VJZgXTfras0kI6Z
5 sryHRpGHUs+gjddWfAap/f9dbtSeDi7dPSW+
Nr6vzwd53N5VadqKd2N2x2CMEUAP
6 cwYw0n6dGCyojN4E5Mo1/rJ1QEn6y9Hvuqb80wpudPxw/phKhax+
VIyDPaH6QMxR
7 LnZvpepc4HnQrzXmie99Bhb/
JfAQv34hoNgJ5HnoUz02n0xCxelBG7rIWUs/VXG
8 BQIDAQAB
9 -----END PUBLIC KEY-----
```

10


```
1 -----BEGIN RSA PRIVATE KEY-----
2 MIIEowIBAAKCAQEaqzzsGK6N9oxQLtcc4zdsygDF6Cruv5qlBHl0
3 0rSzQAk+0By+
4 LchMJ02mQKLSw3moxHhogTLJpKecTusBS/PD/
5 VNYz2gjBLe4Db4jPyT59Ah40bA
6 HI1z5C/PqH82b51TX0+4RqC7MprCdSeeNXQcL5V+
7 10b30GIoyKkw4lCVRrnCclWF
8 5u0E1YnDFjCsdKTW+
9 LcwY0QuR7vq2PHpbeW0m8dJz1Y2zuXTxc27ztqGPcriFrtm
10 7DKV/3Li9g6WsT1AQ0000F+SaiW866+
11 u2zNc9jST2gKzKSYMktcYPHnusblnRzLC
12 pLpB8hPHEkc2ApdZRvoSc+Mq+9kwjUMyXEy/
13 qwIDAQABAoIBAANaLvkQucDA4HT6
14 Sxt7o0qVF0rDRGdF3MMoqM1hMj0nsSsiD0sSh8MhNwb+
15 6Qdgo1gtT0ZwW6u4iEvX
16 N/
17 BHtmeIMS3mSQE3o4fJMvW3oCGipncvlGU6s7Ec6oD09L7copwoKU
18 Bgtyl3dCUd
19 AHdldAP0dmwulV1j/o5nGYjksPdwdVB5w/
20 xnt79FfcAqiYQkShLPUSB9vNGvYc3a
21 xLPupa28dW06ae22/XMTvIdMD73AAMfwTXPwIEkYiQ8L/
22 rALGgKxiXDvEHoGNUjj
23 8oRJJ0I6fflqzWDunf1crs/exPI+wyxCQBsns9W8R/
24 iJjfzgGMgarW7ifz2VeAjI
25 DUhGPLECgYE4bwY49tNu9D8/wsZHqEVuMJZWkyQszW/
26 9VF2jns807feaYkg++H
27 Tax1zIrLjZeyXheEd0ePUCXwp+A+
28 puJfMQDUygyd39GJq5JK0QtjLGNfVslcIbwS
29 7cveAAKHPgtZ/315DceZp0T/
30 etmSIPKfyKfRwXkdT9gc48BJVJCyYbkCgYEAwjJU
31 fBmcife7c7M9iXTgznsptw14l5eWNGUhEtq12ah9B+
32 SW0102fBvmert17B2qqrfD
33 7WZ+
34 0u5LukVKTZsLYbK3d1tYECi8IX3dTEM16tzI07rlidiSLMYmVGB2y
35 rUFEla0i
36 GqToI3os5hjgeb2LFnfCDG2ElXyV3cnEKcfhroMCgYEajlphU+
37 gryEK0GYR1LILy
38 xZxdg49oe8nT1/g2Gd2nt3Tch0moA2/
39 dYrVcgEYTAdLlUAMCrXn6Pa//aM64k+Nz
40 5mJAzr5QHSJ18DXMctdMjmSIBiGDsV6K8tc6w8TUZuMfuUF2PCNg
41 maSgfGeSiKaY
```

```
21 7yAt3hWzz3NZHKNZWzP42jkCgYA/  
    YcntTzamWTLXSnMVQA53lf9BfaYUZCdkJnWq  
22 /  
    7NGvRVB1DusVWB EFZ8eA70z0W0QoLXT8BXXChuxShg8Rf3Ma1YyI  
    KAXdhQhIkFu  
23 0KmKZFEvudpWuwqr0mp tGpRM s/  
    a8m2t8IsKZgbDR00DDGzggNyoggEY7vBP19Xq0  
24 WKXyHQKBgEtL9XgELGf+huemb5xcnkhPEPpfhhGLcJdYa0bK/  
    zvSvHb5KPW/Nq90  
25 qTTDc1roXW9sXJ+  
    Iaz3hUZkCHBLaqFmeqLsXTw9dGI sANjVyxQFiArqLg88H/OP/  
26 1zSh3LNMotIryeDXUPhpfaFJurZU3oGC0vqdPBiAFe3KLSSmn3G5  
27 -----END RSA PRIVATE KEY-----  
28
```

```
1 -----BEGIN RSA PRIVATE KEY-----
2 MIIEpAIBAAKCAQEAssGOMnKA/0q+
  PtM3sCJRhw0jNocYURePnuE82Ahch1wIAYTY
3 NsoKPy0ft3HJiTHNviWC/G6iIJBNTvcznuZz0UwtkVe8Pq/
  iZuK976s/Yg+1SIrT
4 qETTUbwNhmWEL4kjCzI4VEmrjgdnKnZKHDxocSnwZUXDv55HNTpA
  i5CGNwFVQnaK
5 x0RDp3nhMY5vnSb0s2vpAExtKjKeJydhWLAhfJP5qEDxD4q0aJQJ
  yZbXsvT+tmm4
6 QnuF+
  ptGwDmG3xAZA9Lbtk6xe0plUMzQ0KnNnm9CUBEGoOYAVyGWv9ymL
  /ucPIaC
7 DdjW0KR0RuMqqqLeiROGjs3BAomkk548WzduFwIDAQABoIBAAyK
  eIprPNuI6fim
8 sWcwBrUas2UQR1miAaFoA/s4blWIaA4FTpixYJ7DIi/
  fjRlVyMgb6eBavN95le58
9 uB1hteEfSQGRjjpp0NJoQDaYH9lyyLvxaUyC8PACEkToAB7SC446
  74yV63s57eWb
10 ovhNaxlYY5ZXS16uRawikE07biE9r7WVRQIrKesbwMFvDD3T8VkJ
  3Q6m9dAF3uFm
11 En7++v/sEJM1/
  vBFd2QtCGPT9AajqI8XDrfJVEdIfd8iUP243uwSjgCsGZoJyPq
12 X/beLBK+
  BLtrJB0obw9UUukD23NjVaAd6rg13smasCvd9RN9ak2V1ivRudjM
  kG9S
13 JloBKwECgYEATJXiTcWXieqvkw9nCdD943Lxj/
  nZSksYj3fpouEEt0grNNruu6pw
14 igJmNYUA/KUIN1N5pxy0T4x+
  Wgr8HqcUz1yqQsv90FtGpVtTWXs8KlBguzFS2/0
15 1zgJayBe/
  7Fjt1ZW06wdKw5d8a5ZYaPW0LDsrgD0YlYhTCYht2WQ5ncCgYEaw
  WzP
16 DjnV9JdQdy7Z6DLn0n753Akmjvrf+sKvbsW2HDiqMKZg+
  DTbTlRlsZzCbv6AMe4f
17 Ji+3bA5d+
  Scdaq8BtDP8ZniLXW71KJ8CT0zPoGC02vYu0nnIBFEuprJGH/
  BIKCFZ
18 pYAhovnN7vDbNrgT+P1T3/pzydb7ZWpWGF4fWECgYB+
  Fq01r0lER7+qUyUnJisp
19 vWjgwtf7sGo2jEIIfMR1Zgcg9E2n1v6DjyPKAKi0XYAzfFmffff5x
  5vvScKJ6V95Y
```

```
20 vrcNQNke0cZcFKdcSKYu3QX9MRM1UTF7onHili4L0A8liW5dAa3J  
9K480B7y2s7y  
21 ClkZo3RbFGxKmUf+5jKDoQKBgQCal/  
i1iiD7AKVTXKLVLoYSrww3S9wP5sctNW6  
22 V1NCIxgDYjdGqhhN9q6A0qWkSMz1GzjSMHkNaD47kM02LMHT4Wju  
DZJ1zuUq3kve  
23 Tan0qNZj2zb/ja4LpUb//  
KSHimhhiqY33L0Fr9HRfcSoEBgQQkYikD2zT+Xtchyz  
24 gtPmgQKBgQDr57LmPSk5/  
DvcIrQQTwFR6BSR7aizJjgbk72taip94K063Q7RJF3  
25 1TWkjDRJdnh5ZPgF+FC5tn23wBK6hj3rF8bNYMJWLAEVoH2/  
Ks0ZSSmQjRPEJIPK  
26 Grn00ue1Irs4fl065ptjZtw8HgKitE9sle1vIgwpUKQXyFrFXAit  
kg==  
27 -----END RSA PRIVATE KEY-----  
28
```

```
1 -----BEGIN RSA PRIVATE KEY-----
2 MIIEpQIBAAKCAQEAmwTyC9d846+33cLYUSkcd+x9i/3j292+
v00wUp1WWA/ceJv
3 VbtkyieVAPRmW9o8ow12e67blp0e5TKtksjR0I7AnVVT2zJgV4Xb
fj/Jqu6x5CNd
4 jVA4vwRnYVp+hEqv907A2VJZgXTfras0kI6ZsryHRpGHUs+
gjddWfAap/f9dbtSe
5 Di7dPSW+
Nr6vzwd53N5VadqKd2N2x2CMEUAPcwYw0n6dGCyojN4E5Mo1/
rJ1QEn6
6 y9Hvuqb80wpudPxw/phKhax+
VIyDPaH6QMxRLnZvpepc4HnQrzXmie99Bhb/JFAQ
7 v34hoNgJ5HnoUz02n0xTCxelBG7rIWUs/
VXGBQIDAQABAOIBAQCKPl8ykLu1W2LN
8 cuLZbv2fGvhnnWPiUdfASVnVtyQKES4LxH5173GTb0G+
dAn0dhF3vzLob0Ukq0Q0
9 pPxWywQ5uweq1+lwizmGAmA4PRMgv+
kt1xfGS6yN5ou75aTgV7vjldP6s1uBeaEy
10 0/
HkeAFB3lwyCXi3oXsAYIXWWsa1G4n2lEj5FlhWLnuJ2KSEA6PejF
RopB4GzA7m
11 xYyZre4wKN7oTiSke+
Pm2xYVMz hwxDG0PQdYZQtyle0ZU8n2E8NxAGrh4KkSNVcP
12 9yknoVYz6Lq85lABziMu hmHnnEGs3r2TRkp29aWJ1IQYjX7/
UFWEBdqrIm9ePkdp
13 vX+
S6lABAoGBA00jqqqGrAp21s0f2LXh2lsiq5pnGBI2KqLXi05XfeJM
H0vN9MWt/
14 tMgDtTeD7gbYC2pXIkpFuyVQjP100l/
R6Hvm3BzaKwEAtXbsfdmzTxqfs7HI8SXR
15 pwKWqr3m6oFOE+
EuNBhcbfucmhB3Wg5lseSDHcRxTLIuvjl1bhZzoqBAoGBANqk
16 +32nhDmou11kXMA s7ipv49oc6T6n2cs1Gy/
Fbqq1lq4TgDDVgoKwznYB1habFy+E
17 e3qCPcu8JVXu0so+
L0j21yQ1WlC00lIf0bMgqLCpZjzMqa9upDE304ogJqQJDQk2
18 g/
LG01fQJtafe3QeoHeTZ9w0buk99Qv8guE0tVGFAoGBAMS6pzY8kn
A1m4N8L2PS
19 KRGXEN+
IUFr6oD9AGvq1PIDkWMAhS9p5bYUGH1Cx Bb6Ia6ULUyeUR95BtPc
```

```
19 9wU0a
20 HW8m3sdYjJ27PRhf3YuM+SorJqLY4/8pJsqH51ti+
    vtwvKF4yrDbAHnYpxuboHr4
21 eiRT Rc6JXwE6lMuR5ZgClQsBAoGAI8puAJuzYVzljtwm8q5oLjoy
    qjmhVMhVNpZy
22 5NcEzp z7FXPLwDKzM oG0ynJygTDSEs01CVDYnMkns3FT3ldflioR
    /bNeHWfjRB4o
23 a9Ik ywZv3fQCst0Bs6zXy/yHV sLEh4WNA+jYH7/
    LG8bvhoqc6fYPQk56iWPDATtM
24 kVq/A6ECgYE AuXIqXKi iU+xElmI/3F0aMQDk5z+
    Rodrc8hsUM7BehsmrjerHPK1i
25 qzel3EYEzxdt4vLhgHBgpaA1I91duYQcWENDBUZ134wNx7Sf5s4Q
    i/hrlo4obVA6
26 lEwGgQuHE6XQWH10mouAlp4s4snfRreXurf0MYgMEVlGXk0sDAsn
    xm4=
27 -----END RSA PRIVATE KEY-----
28
```

```
1 # coding=utf-8
2 from setuptools import setup, find_packages # type
3 : ignore
4
5     name="settle-server",
6     version="0.1.0",
7     py_modules=[ "endpoint" ],
8     entry_points="""
9     [console_scripts]
10    settle-server=endpoint:settle_server
11    """
12 )
13
```



```
1 # coding=utf-8
2 from dataclasses import dataclass
3
4 import click
5
6
7 @dataclass
8 class User:
9     name: str
10    email: str
11    modulus: str
12    pub_exp: str
13    password: str = "default"
14    id: int = 0
15
16    def __str__(self):
17        return f"""\nName:\t{self.name}
18 Email:\t{self.email}
19 Modulus:\t{self.modulus},
20 Public Exponent:\t{self.pub_exp}
21 """
22
23
24 @dataclass
25 class Group:
26    id: int
27    name: str
28    password: str
29
30    def __repr__(self):
31        return f"Group(name={self.name}, id={self.id
32 })"
33
34    def __str__(self):
35        return f"Group:\n\tName: {self.name}, ID = {
36 self.id}"
37
38 @dataclass
39 class GroupLink:
40     id: int
```

```
40     group_id: int
41     usr_id: int
42
43
44 @dataclass
45 class GroupList:
46     groups: list[Group]
47
48     def __str__(self):
49         out = ""
50         for group in self.groups:
51             out += f"{str(group)}\n"
52
53     return out
54
55
56 @dataclass
57 class PrettyTransaction:
58     id: int
59     group: int
60     amount: int
61     time: str
62     reference: str
63     other: str
64     verified: bool
65
66     def secho(self):
67         click.secho("\n----")
68         click.secho(f"Transaction ID = {self.id}\n")
69         click.secho(f"Group: {self.group}", bold=True)
70         click.secho(f"{self.other}, £{(int(self.
71             amount) / 100):.2f}", fg="blue")
72         click.secho(self.reference)
73         click.secho(f"at {self.time}")
74
75         if self.verified:
76             click.secho("Verified: True\n", fg="
77             green")
78
79         else:
80             click.secho("Verified: False\n", fg="red")
```

```
77 " , blink=True, bold=True)
78
79
80 @dataclass
81 class PrettyList:
82     src_list: list[PrettyTransaction]
83     dest_list: list[PrettyTransaction]
84
85     def __bool__(self):
86         return True if (self.src_list or self.
87         dest_list) else False
88
89     def __repr__(self):
90         return f"{self.src_list}"
91
92     def secho(self):
93         for trn in self.src_list:
94             trn.secho()
95         for trn in self.dest_list:
96             trn.secho()
97
98 @dataclass
99 class Signature:
100     transaction_id: int
101     signature: str # store as hex
102     origin: str # src or dest
103
```



```
1 # coding=utf-8
2
3 from marshmallow import Schema, fields, post_load
4
5 import models # type: ignore
6 import src.transactions.ledger
7 import src.transactions.transaction
8
9
10 class UserSchema(Schema):
11     id = fields.Int()
12     name = fields.Str()
13     email = fields.Email()
14     modulus = fields.Str()
15     pub_exp = fields.Str()
16     password = fields.Str()
17
18     @post_load
19     def make_user(self, data, **kwargs):
20         return models.User(**data)
21
22
23 class GroupSchema(Schema):
24     id = fields.Int()
25     name = fields.Str()
26     password = fields.Str()
27
28     @post_load
29     def make_group(self, data, **kwargs):
30         return models.Group(**data)
31
32
33 class GroupLinkSchema(Schema):
34     id = fields.Int()
35     group_id = fields.Int()
36     usr_id = fields.Int()
37
38     @post_load
39     def make_group_link(self, data, **kwargs):
40         return models.GroupLink(**data)
41
```

```
42
43 class GroupListSchema(Schema):
44     groups = fields.List(fields.Nested(GroupSchema
        ()))
45
46     @post_load
47     def make_group_list(self, data, **kwargs):
48         return models.GroupList(**data)
49
50
51 # Transaction schemas
52
53
54 class TransactionSchema(Schema):
55     src = fields.Int()
56     dest = fields.Int()
57     amount = fields.Int()
58     src_pub = fields.Str()
59     dest_pub = fields.Str()
60     ID = fields.Int()
61     reference = fields.Str()
62     time = fields.Str()
63     signatures = fields.Dict()
64     group = fields.Int()
65
66     @post_load
67     def make_transaction(self, data, **kwargs):
68         return src.transactions.transaction.
    Transaction(**data)
69
70
71 class PrettyTransactionSchema(Schema):
72     id = fields.Int()
73     other = fields.Str()
74     group = fields.Int()
75     amount = fields.Int()
76     time = fields.Str()
77     reference = fields.Str()
78     verified = fields.Bool()
79
80     @post_load
```

```
81     def make_pretty_transaction(self, data, **  
82         kwargs):  
83             return models.PrettyTransaction(**data)  
84  
85 class PrettyListSchema(Schema):  
86     src_list = fields.List(fields.Nested(  
87         PrettyTransactionSchema()))  
88     dest_list = fields.List(fields.Nested(  
89         PrettyTransactionSchema()))  
90  
91     @post_load  
92     def make_pretty_list(self, data, **kwargs):  
93         return models.PrettyList(**data)  
94  
95 class SignatureSchema(Schema):  
96     transaction_id = fields.Int()  
97     signature = fields.Str() # store as hex  
98     origin = fields.Str() # src or dest  
99  
100    @post_load  
101    def make_signature(self, data, **kwargs):  
102        return models.Signature(**data)
```



```
1 # coding=utf-8
2 import os
3
4 import click
5 from flask import Flask, g
6 from flask_restful import Resource, Api, abort  # type: ignore
7
8 from src.server.processes import get_db
9 from src.server.resources import (
10     Group,
11     User,
12     UserGroupBridge,
13     PrettyTransaction,
14     TransactionSigVerif,
15     Simplifier,
16     GroupDebt,
17     SignableTransaction,
18 )
19
20 app = Flask(__name__)
21 api = Api(app)
22
23
24 @app.teardown_appcontext
25 def close_connection(exception):
26     """Closes db if sudden error"""
27     db = getattr(g, "_database", None)
28     if db is not None:
29         db.close()
30
31
32 @click.group()
33 def settle_server():
34     ...
35
36
37 @click.option("-d", "--debug", is_flag=True, default=False)
38 @click.option("-h", "--host", default="127.0.0.1")
39 @settle_server.command()
```

File - /home/tcassar/projects/settle/src/server/endpoint.py

```
40 def start(host, debug):
41     os.chdir("/home/tcassar/projects/settle")
42     app.run(debug=debug, host=host)
43     db = get_db()
44
45
46 api.add_resource(Group, "/group/<int:id>", "/group")
47
48 api.add_resource(PrettyTransaction, "/transaction",
49                  "/transaction/<string:email>")
50
51 api.add_resource(User, "/user/<string:email>", "/user")
52
53 api.add_resource(
54     UserGroupBridge, "/group/<int:id>/<string:email>",
55     "/group/<string:email>")
56
57 api.add_resource(
58     TransactionSigVerif, "/transaction/auth/<int:id>",
59     "/transaction/auth/")
60
61 api.add_resource(Simplifier, "/simplify/<int:gid>")
62
63 api.add_resource(GroupDebt, "/group/debt/<int:id>")
64
65 api.add_resource(SignableTransaction, "/transaction/
settle/<int:t_id>", "/transaction/signable/<int:id>")

```

```
1 # coding=utf-8
2 import sqlite3
3
4 from flask import g
5
6 import src.server.models as models
7 import src.transactions.transaction
8 from src.crypto import keys as keys
9
10 DATABASE = "/home/tcassar/projects/settle/settle_db.sqlite"
11
12
13 class ResourceNotFoundError(Exception):
14     ...
15
16
17 def get_db():
18     """Returns current database connection"""
19     db = getattr(g, "_database", None)
20     if db is None:
21         # connect
22         db = g._database = sqlite3.connect(DATABASE)
23
24     return db
25
26
27 def build_args(data_from_cursor: list | tuple) -> list:
28     """Given a ROW OF PARAMETERS FROM DB will return
29     list as args"""
30     if args := data_from_cursor:
31         args = [item for item in args]
32         return args
33     else:
34         raise ResourceNotFoundError(
35             "Database error: failed to build schema
36             of object as nothing was retrieved"
37         )
```

```
38 def build_pretty_transactions(
39     src_sql: str, dest_sql: str, cursor: sqlite3.
40     Cursor, args: list
40 ) -> models.PrettyList:
41     """Returns pretty list of unchecked transactions
41 """
42     # TODO: add transaction verification
43
44     if type(args) is not list:
45         args = list(args)
46
47     src_transactions: list[models.PrettyTransaction]
47     ] = []
48     dest_transactions: list[models.PrettyTransaction]
48     ] = []
49
50     src_data = cursor.execute(src_sql, args)
51
52     for row in src_data.fetchall():
53         pretty = models.PrettyTransaction(*
53         build_args(row), False)
54         print(f"Checking id {pretty.id}")
55         verify_pretty(pretty, cursor)
56         src_transactions.append(pretty)
57
58     dest_data = cursor.execute(dest_sql, args)
59
60     for row in dest_data.fetchall():
61         pretty = models.PrettyTransaction(*
61         build_args(row), False)
62         print(f"Checking id {pretty.id}")
63         verify_pretty(pretty, cursor)
64         dest_transactions.append(pretty)
65
66     return models.PrettyList(src_transactions,
66     dest_transactions)
67
68
69 def get_verified_transaction_by_id(
70     id: int, cursor: sqlite3.Cursor
71 ) -> src.transactions.transaction.Transaction:
```

```

72     """Gets transaction object from id
73     raises ResourceNotFoundError if nothing is
    returned"""
74
75     sql = """SELECT src_id, dest_id, k.n, k.e, k2.n
    , k2.e, amount, transactions.id, reference,
    time_of_creation, src_sig, dest_sig, g.id FROM
    transactions
76 JOIN keys k ON transactions.src_key = k.id
77 JOIN keys k2 ON transactions.dest_key = k2.id
78 JOIN pairs p ON transactions.pair_id = p.id
79 JOIN groups g ON transactions.group_id = g.id
80 WHERE transactions.id = ?;
81 """
82
83     raw_transaction_data = cursor.execute(sql, [id
    ]).fetchone()
84     if not raw_transaction_data:
85         raise ResourceNotFoundError(f"No
transaction with ID={id}")
86
87     else:
88         src_key_data = raw_transaction_data[2:4]
89         dest_key_data = raw_transaction_data[4:6]
90         transaction_data = list(
raw_transaction_data[:2]) + list(
91             raw_transaction_data[6:])
92         )
93
94         (
95             src_id,
96             dest_id,
97             amount,
98             t_id,
99             ref,
100            time,
101            src_sig,
102            dest_sig,
103            group,
104        ) = transaction_data
105

```

File - /home/tcassar/projects/settle/src/server/processes.py

```
106     print(
107         f"amount: {amount}\n"
108         f"tran_id: {t_id}\n"
109         f"ref: {ref}\n"
110         f"time: {time}\n"
111         f"src_sig: {src_sig}\n"
112         f"dest_sig: {dest_sig}\n"
113         f"group: {group}\n"
114     )
115
116     # build keys
117     # convert hex -> ints
118     src_key_data = map(lambda x: int(x, 16),
119                         src_key_data)
120     dest_key_data = map(lambda x: int(x, 16),
121                         dest_key_data)
122
123     # load to key loaders
124     src_ldr = keys.RSAKeyLoaderFromNumbers()
125     dest_ldr = keys.RSAKeyLoaderFromNumbers()
126
127     src_ldr.load(*src_key_data)  # type: ignore
128     dest_ldr.load(*dest_key_data)
129
130     # add keys to transaction data in the right
131     # place so that they can be unpacked as positional
132     # arguments
133     transaction_data.insert(3, dest_ldr.pub_key)
134     transaction_data.insert(3, src_ldr.pub_key)
135
136     # build signatures dict for transaction
137
138     signatures = {}
139
140     for sig, notary in zip([src_sig, dest_sig],
141                           [src_id, dest_id]):
142         if sig:
143             signatures[notary] = int(sig, 16)
```

File - /home/tcassar/projects/settle/src/server/processes.py

```
140         # remove signatures from transaction data,
141         replace w dict
142         transaction_data.remove(src_sig)
143         transaction_data.remove(dest_sig)
144
145         transaction_data.insert(-1, signatures)
146
147         # return transaction
148         return src.transactions.transaction.
149             Transaction(*transaction_data)
150
151
152     def user_exists(email: str, cursor: sqlite3.Cursor
153 ) -> bool:
154         return not not cursor.execute(
155             """SELECT COUNT(*) FROM users WHERE email
156 = ?""", [email]
157         ).fetchone()[0]
158
159
160
161
162     def group_exists(id: int, cursor: sqlite3.Cursor
163 ) -> bool:
164         return not not cursor.execute(
165             """SELECT COUNT(*) FROM groups WHERE id
166 = ?""", [id]
167         ).fetchone()[0]
168
169
170
171     def transaction_to_pretty(emails, transaction,
172     verified):
173         pretty = models.PrettyTransaction(
174             transaction.ID,
175             transaction.group,
176             transaction.amount,
177             transaction.time,
178             transaction.reference,
179             f"{emails[0]} -> {emails[1]}",
180             verified,
181         )
182
183         return pretty
```

```

174
175 def verify_pretty(
176     pretty: models.PrettyTransaction, cursor:
177     sqlite3.Cursor
178 ) -> models.PrettyTransaction:
179     """Update the verification status of pretty
180     depending on signatures"""
181     try:
182         t = get_verified_transaction_by_id(pretty.
183             id, cursor)
184         t.verify()
185         pretty.verified = True
186     except src.transactions.transaction.
187         VerificationError:
188             pretty.verified = False
189             print(f"Signature of {pretty.id} invalid")
190
191     return pretty
192
193
194     # check pair exists; append if not
195     pair_id: int = cursor.execute(
196         """SELECT pairs.id FROM pairs WHERE src_id
197         = ? and dest_id = ?""",
198         [
199             transaction.src,
200             transaction.dest,
201         ],
202         ).fetchone()[0]
203
204     if not pair_id:
205         cursor.execute(
206             """INSERT INTO pairs (src_id, dest_id)
207             VALUES (?, ?)""",
208             [
209                 transaction.src,

```

```
208                 transaction.dest,
209                 ],
210             )
211         get_db().commit()
212
213     # get key ids of users
214     key_ids = cursor.execute(
215         """SELECT keys.id FROM keys
216             JOIN users u ON keys.id = u
217             .key_id
218             JOIN users u2 ON keys.id =
219             u2.key_id
220             WHERE u.id = ? OR u2.id
221             = ?""",
222         [transaction.src, transaction.dest],
223         ).fetchall()
224
225     sql = """INSERT INTO transactions
226         (pair_id, group_id, amount, src_key, dest_key,
227         reference, time_of_creation)
228         VALUES (?, ?, ?, ?, ?, ?, ?, ?)"""
229
230     print(key_ids[0][0], key_ids[1][0])
231
232     # append unsigned transactions
233     cursor.execute(
234         sql,
235         [
236             pair_id,
237             transaction.group,
238             transaction.amount,
239             key_ids[0][0],
240             key_ids[1][0],
241             transaction.reference,
242             transaction.time
243         ],
244     )
245
246     get_db().commit()
```



```

1 # coding=utf-8
2 import json
3
4 from flask import request
5 from flask_restful import Resource, abort # type: ignore
6
7 import src.transactions.transaction as transactions
8 import src.transactions.ledger as ledgers
9 from src.server import models as models, schemas as schemas, processes as processes
10
11
12 # Resources
13
14
15 class Group(Resource):
16     def get(self, id: int):
17         cursor = processes.get_db().cursor()
18         # check group exists and
19         get_group = """SELECT id, name, password
FROM groups
          WHERE groups.id = ?"""
20
21         try:
22             group_data = cursor.execute(get_group, [id])
23             group_data = group_data.fetchall()
24             # create group object
25             group = models.Group(*processes.
26                 build_args(*group_data))
27         except IndexError:
28             abort(404, message="Group ID does not
exist")
29         group = "" # type: ignore
30         except TypeError as te:
31             abort(404, message=f"Group data invalid
; {te}")
32         group = "" # type: ignore
33         # create group schema
34         schema = schemas.GroupSchema()
35         return schema.dump(group), 200

```

```

35
36     def post(self):
37
38         # print(request.json)
39         schema = schemas.GroupSchema()
40         group = schema.load(request.json)
41
42         cursor = processes.get_db().cursor()
43
44         cursor.execute(
45             """INSERT INTO groups (name, password)
46             VALUES (?, ?)""",
47             [group.name, group.password],
48         )
49
50         processes.get_db().commit()
51
52         return f"Created group ID={cursor.lastrowid}"
53         named {group.name}", 201
54
55 class User(Resource):
56     def get(self, email: str):
57         # query db for all users
58         cursor = processes.get_db().cursor()
59
60         query = """
61             SELECT users.name, users.email,
62             keys.n, keys.e, users.password, users.id
63             FROM users, keys
64             WHERE email = ? AND keys.id = users
65             .key_id;
66             """
67
68         # only return one usr, so unpack first into
69         # usr_data
70         try:
71             usr_data: list = cursor.execute(query, [
72                 email]).fetchone()
73         except IndexError:
74             # returned blank info

```

```

70             return "User data not found", 404
71
72         # use data to build user class
73         # use schema to convert to json
74
75     try:
76         # make sure the requested user exists
77         usr = models.User(*[item for item in
78             usr_data])
79         schema = schemas.UserSchema()
80     except TypeError:
81         # didn't have required arguments to
82         # build usr
83         return "User data invalid", 404
84
85     def post(self):
86
87         cursor = processes.get_db().cursor()
88
89         # now is a user object
90         schema = schemas.UserSchema()
91         usr = schema.load(request.json)
92
93         query = """SELECT users.id FROM users WHERE
94             email = ?
95             """
96
97         exists = cursor.execute(query, [usr.email])
98         if exists.fetchall():
99             abort(409, message="User already exists")
100
101         # add user to db
102
103         keys_query = """INSERT INTO keys (n, e)
104                         VALUES (?, ?)"""
105
106         users_query = """INSERT INTO users (NAME,
107             EMAIL, PASSWORD, KEY_ID)

```

```

106                                     VALUES (?, ?, ?, ?, ?)"""
107
108         cursor.execute(keys_query, [usr.modulus,
109                         usr.pub_exp])
110
111         key_id = cursor.lastrowid
112         cursor.execute(users_query, [usr.name, usr.
113                         email, usr.password, key_id])
114         processes.get_db().commit()
115
116
117     class UserGroupBridge(Resource):
118         """For handling users connections to groups
119         POST will add user to group
120         GET will get all groups associated with user"""
121
122         def post(self, id: int, email: str):
123             # assumes these things already exist as
124             # they have been validated by client already
125             uid_sql = """SELECT id FROM users WHERE
126 email = ?"""
127
128             glink_sql = """INSERT INTO group_link (
129 group_id, usr_id)
130                                     VALUES (?, ?)""" # group
131             # id then user id
132
133             cursor = processes.get_db().cursor()
134             uid = cursor.execute(uid_sql, [email]).fetchone()[0]
135
136             cursor.execute(glink_sql, [id, uid])
137
138             processes.get_db().commit()
139
140             glink_data = cursor.execute(
141                 """SELECT * from group_link WHERE id
142 = ?""", [cursor.lastrowid]
143             )

```

```

139
140         try:
141             print("making group")
142             glink = models.GroupLink(*processes.
143             build_args(glink_data.fetchone()))
144         except processes.ResourceNotFoundError as
145             re:
146                 return 404, f"{re}, failed"
147
148             schema = schemas.GroupLinkSchema()
149
150     def get(self, email: str):
151         """Return all groups that a user is part of
152         """
153
154         sql = """SELECT g.id, g.name, g.password
155             FROM users
156                 JOIN group_link gl ON users.id = gl.usr_id
157                 JOIN groups g ON g.id = gl.group_id
158                 WHERE users.id
159                     =
160                         (
161                             SELECT users.id FROM users
162                             WHERE email = ?
163                         );
164         """
165
166         cursor = processes.get_db().cursor()
167         group_data = cursor.execute(sql, [email])
168         groups: list[models.Group] = []
169
170         for row in group_data.fetchall():
171             groups.append(models.Group(*processes.
172             build_args(row)))
173
174         groups_obj = models.GroupList(groups)
175         groups_schema = schemas.GroupListSchema()
176
177         return groups_schema.dump(groups_obj), 200
178

```

```

175
176 class PrettyTransaction(Resource):
177     def get(self, email: str):
178         """Gets a user's unsigned transactions"""
179
180         cursor = processes.get_db().cursor()
181
182         if not processes.user_exists(email, cursor):
183             print("not found")
184             return f"User by email {email} not found", 404
185
186         src_sql = """
187             SELECT transactions.id,
188                 group_id, amount, reference, time_of_creation, u2.
189                 email FROM transactions
190                     INNER JOIN pairs p on p.id =
191                         transactions.pair_id
192                             INNER JOIN users u on u.id = p.
193                         src_id
194                             INNER JOIN users u2 on u2.id = p.
195                         dest_id
196                             WHERE transactions.src_settled = 0
197                         AND u.email = ?
198
199         """
200
201         dest_sql = """
202             SELECT transactions.id, group_id, amount,
203                 reference, time_of_creation, u2.email FROM
204                 transactions
205                     INNER JOIN pairs p on p.id =
206                         transactions.pair_id
207                             INNER JOIN users u on u.id = p.
208                         dest_id
209                             INNER JOIN users u2 on u2.id = p.
210                         src_id
211                             WHERE transactions.src_settled = 0
212                         AND u.email = ?"""
213
214         pretty_list = processes.
215             build_pretty_transactions(

```

```

201             src_sql, dest_sql, cursor, [email]
202         )
203
204     pretty_list_schema = schemas.
205     PrettyListSchema()
206     if not pretty_list:
207         return "No open transactions", 200
208     else:
209         return pretty_list_schema.dump(
210             pretty_list), 200
211
212
213     # note: IDs are ints
214
215     # load transaction object into schema from
216     # request
217     trn_json = request.json
218     trn_schema = schemas.TransactionSchema()
219     transaction = trn_schema.make_transaction(
220         trn_json)
221
222     # check that both people involved in
223     # transaction are members of transaction group
224     sql = """SELECT count(*) FROM groups
225             INNER JOIN group_link gl ON groups
226             .id = gl.group_id
227             INNER JOIN users u
228             INNER JOIN users u2
229             WHERE group_id = ? AND u.id = ?
230             AND u2.id = ?
231             """
232
233     users_in_group = cursor.execute(
234         sql, [transaction.group, transaction.
235             src, transaction.dest]
236     )
237     if users_in_group.fetchone()[0] == 0:
238         return f"Users are not both members of
239         group {transaction.group}", 403

```

```
233
234         insert_to_pairs = """INSERT INTO pairs (
235             src_id, dest_id)
236                 VALUES (?, ?) ON
237                 CONFLICT DO NOTHING """
238
239         cursor.execute(insert_to_pairs, [
240             transaction.src, transaction.dest])
241         processes.get_db().commit()
242
243         # get relevant info
244         pair_id = cursor.execute(
245             """SELECT pairs.id FROM pairs
246                 WHERE src_id = ? AND dest_id
247                 = ?""",
248                 [transaction.src, transaction.dest],
249                 ).fetchone()[0]
250
251         key_id_query = """SELECT keys.id FROM keys
252                         JOIN users u on keys.id
253                         = u.key_id
254                         WHERE u.id = ?"""
255
256         src_key_id = cursor.execute(key_id_query, [
257             transaction.src]).fetchone()[0]
258         dest_key_id = cursor.execute(key_id_query,
259             [transaction.dest]).fetchone()[0]
260
261         cursor.execute(
262             """INSERT INTO transactions
263                 (pair_id, group_id, amount, src_key,
264                 dest_key, reference, time_of_creation)
265                 VALUES (?, ?, ?, ?, ?, ?, ?, ?)""",
266             [
267                 pair_id,
268                 transaction.group,
269                 transaction.amount,
270                 src_key_id,
271                 dest_key_id,
272                 transaction.reference,
273                 transaction.time,
```

File - /home/tcassar/projects/settle/src/server/resources.py

```

300     id
301             WHERE transactions.id = ?
302             """
303             [id],
304         )
305         .fetchone()
306     )
307
308     # build pretty transaction to send back to
309     # the user
310
311     pretty = processes.transaction_to_pretty(
312         emails, transaction, verified)
313
314     schema = schemas.PrettyTransactionSchema()
315
316     def patch(self):
317         """Append a signature to a transaction"""
318
319         schema = schemas.SignatureSchema()
320         sig = schema.make_signature(request.json)
321         if request.json is None:
322             return "Invalid Request", 404
323
324         if sig.origin == "dest":
325             sql = """ UPDATE transactions
326                     SET dest_sig = ?
327                     WHERE id = ?"""
328         elif sig.origin == "src":
329             sql = """UPDATE transactions
330                     SET src_sig = ?
331                     WHERE id = ?"""
332         else:
333             return (
334                 "Signature does not originate from
335                 one of the parties in this transaction",
336                 403,
337             )

```

```
338         cursor = processes.get_db()
339         cursor.execute(sql, [sig.signature, sig.
340             transaction_id])
341         processes.get_db().commit()
342
343         return "Successfully added signature to
344             transaction", 201
345
346     class Simplifier(Resource):
347         def post(self, gid: int):
348             """Actually settle the group, return ledger
349                 schema, 201 if succeeded"""
350
351         cursor = processes.get_db()
352
353         # build full transactions of group
354         # get list of IDs required
355
356         print(gid)
357
358         ids = cursor.execute(
359             """SELECT transactions.id from
360                 transactions
361                     WHERE group_id = ?
362                     AND src_settled = 0 AND dest_settled = 0""",
363                     [gid],
364                     ).fetchall()
365
366         unfiltered_transactions: list[transactions.
367             Transaction] = []
368         for id in ids:
369             unfiltered_transactions.append(
370                 processes.get_verified_transaction_by_id(*id,
371                     cursor)) # type: ignore
372
373         # build ledger
374         ledger = ledgers.Ledger()
375         for transaction in unfiltered_transactions:
376             ledger.append(transaction)
```

```

371
372             # simplify debt system
373         try:
374             ledger.simplify_ledger()
375         except ledgers.NoFurtherSimplifications:
376             return (
377                 "No changes made to debt structure
- heuristic did not find anywhere to simplify",
378                     202,
379             )
380         except ledgers.VerificationError:
381             return "Couldn't simplify group -
unverified transactions in group", 403
382
383         # mark old transactions as settled
384
385         # otherwise, group debt is now simplified
386         # thus, mark off all old transactions and
push in new ones
387         cursor.execute(
388             """UPDATE transactions
389             SET src_settled = 1, dest_settled = 1
390             WHERE group_id = ? """,
391             [gid],
392         )
393
394         # push transactions to db
395         for transaction in ledger.ledger:
396             transaction.group = gid
397             processes.push_transaction(transaction
, cursor)
398
399         return 'success', 201
400
401
402 class GroupDebt(Resource):
403     def get(self, id):
404         """Return open transactions in a group"""
405
406         cursor = processes.get_db()
407

```

```

408         sql = """SELECT transactions.id, group_id,
409             amount, time_of_creation, reference, u.email, u2.
410             email
411                 FROM transactions
412                     INNER JOIN pairs p on p.id =
413                         transactions.pair_id
414                             INNER JOIN users u on u.id = p.
415                               src_id
416                               INNER JOIN users u2 on u2.id =
417                                 p.dest_id
418                                     WHERE group_id = ? AND (
419                                         transactions.src_settled = 0 OR transactions.
420                                         dest_settled = 0);
421                                         """
422
423     trns: list[models.PrettyTransaction] = []
424     for row in cursor.execute(sql, [id]).fetchall():
425         row = list(row)
426         dest = row.pop()
427         src = row.pop()
428         row.append(f"{src} -> {dest}")
429
430         pretty = models.PrettyTransaction(*
431             processes.build_args(row), False)
432         pretty = processes.verify_pretty(pretty
433             , cursor)
434
435         trns.append(pretty)
436
437     schema = schemas.PrettyListSchema()
438     return schema.dump(models.PrettyList(trns
439         , [])), 200
440
441
442 class SignableTransaction(Resource):
443     def get(self, id):
444         """Returns a fully signable transaction
445         object"""
446
447     try:

```

```

437         transaction = processes.
438         get_verified_transaction_by_id(
439             id, processes.get_db().cursor()
440         )
440     except processes.ResourceNotFoundError as
441         rnfe:
442             return str(rnfe), 404
443
443     schema = schemas.TransactionSchema()
444     return schema.dump(transaction), 200
445
446     def patch(self, t_id):
447         cursor = processes.get_db()
448         email: dict = json.loads(request.json)[
449             'email']
450
450         # determine if user is src or dest
451         emails = cursor.execute("""SELECT u.email,
452             u2.email FROM transactions
453             JOIN pairs p ON transactions.pair_id = p.id
454             JOIN users u ON p.src_id = u.id
455             JOIN users u2 ON p.dest_id = u2.id
455             WHERE transactions.id = ? """, [t_id]).
456         fetchone()
457
457         if email == emails[0]:
458             # usr is src thus append src_settled
459             sql = """UPDATE transactions SET
460                 src_settled = 1 WHERE id = ?"""
460         elif email == emails[1]:
461             sql = """UPDATE transactions SET
461                 dest_settled = 1 WHERE id = ?"""
462         else:
463             return f'Email provided is not involved
463             in transaction {t_id}', 403
464
465         cursor.execute(sql, [t_id])
466         cursor.commit()
467

```

```
1 # coding=utf-8
2 import logging
3 import sys
4 from dataclasses import dataclass
5 from typing import Callable
6
7 from ordered_set import OrderedSet
8
9 # initialise logger
10 import src.simplify.graph_objects
11 from src.simplify import base_graph as graphs
12
13 logging.basicConfig(stream=sys.stdout, encoding="utf-8", level=logging.DEBUG)
14
15
16 # typing
17 disc_map = dict[
18     src.simplify.graph_objects.Vertex, src.simplify.
19     graph_objects.Vertex | bool
20 ]
21 """disc_map used to track which nodes have been
22 discovered; {node: discovered?}"""
23
24 prev_map = dict[
25     src.simplify.graph_objects.Vertex, src.simplify.
26     graph_objects.Vertex | None
27 ]
28 """prev_map used to track a path through a graph;
29 stored as {node: comes_from_node}.
30 If node has no links (i.e. is start node), value is
31 None"""
32
33 class SearchError(Exception):
34     def __init__(self, text, node: src.simplify.
35     graph_objects.Vertex):
36         super(SearchError, self).__init__(text, node
37     )
38         self.node = node
```

```

34
35
36 def void(
37     current: src.simplify.graph_objects.Vertex,
38     neighbour: src.simplify.graph_objects.Vertex,
39 ) -> None:
40     """Placeholder for a plain bfs; allows adding
41     functionality such as a maxflow along each edge
42     during a BFS"""
43     pass
44
45
46
47
48
49
50
51
52 @dataclass(init=False)
53 class BFSQueue:
54     def __init__(self, *args: src.simplify.
55                 graph_objects.Vertex):
56         self.data: OrderedSet[src.simplify.
57                 graph_objects.Vertex] = OrderedSet(args)
58
59     def __str__(self):
60         str_ = ""
61         for datum in self.data:
62             str_ += str(datum).upper()
63         return str_
64
65     def enqueue(self, *args: src.simplify.
66                 graph_objects.Vertex):
67         for v in args:
68             graphs.GenericDigraph.sanitize(v)
69             self.data.append(v)
70
71     def dequeue(self):

```

```
69         return self.data.pop(0)
70
71     def is_empty(self) -> bool:
72         return not len(self.data)
73
74
75 class Path:
76     """Namespace for graph operations to do with
77     walking through graph"""
78
79     """Shortest path code: returns list[Vertex],
80     hops along a path"""
81
82     @staticmethod
83     def build_bfs_structs(
84         graph: graphs.GenericDigraph,
85         src: None | src.simplify.graph_objects.
86         Vertex = None,
87         ) -> tuple[BFSQueue, disc_map, prev_map]:
88         """Helper function to initialise prev_map,
89         disc_map and bfs queue;
90         if source is passed then queue initialised
91         with src"""
92         queue = BFSQueue()
93         disc: disc_map = {node: False for node in
94             graph.nodes()}
95         prev: prev_map = {node: None for node in
96             graph.nodes()}
97
98         if src:
99             queue.enqueue(src)
100            else:
101                # get 'first' item from graph
102                # n = len(graph.graph)
103                # start = list(graph.graph.keys())[
104                    random.randint(0, n - 1)]
105                # queue.enqueue(start)
106                start = next(iter(graph.graph))
107                logging.debug(f"starting from {start}")
108
109                # queue.enqueue(next(iter(graph.graph)))
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1490
1491
1492
1493
1494
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1590
1591
1592
1593
1594
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1690
1691
1692
1693
1694
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1790
1791
1792
1793
1794
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1890
1891
1892
1893
1893
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1990
1991
1992
1993
1994
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2087
2088
2089
2090
2091
2092
2093
2094
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2187
2188
2189
2190
2191
2192
2193
2194
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2287
2288
2289
2290
2291
2292
2293
2294
2294
2295
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2387
2388
2389
2390
2391
2392
2393
2394
2394
2395
2396
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2477
2478
2479
2480
2481
2482
2483
2484
2485
24
```

```

101    ))))
102
103        return queue, discovered, previous
104
105    @staticmethod
106    def shortest_path(
107        graph: graphs.GenericDigraph,
108        source: src.simplify.graph_objects.Vertex,
109        sink: src.simplify.graph_objects.Vertex,
110        neighbours: Callable,
111    ) -> list[src.simplify.graph_objects.Vertex]:
112        """
113            Uses a recursive implementation of BFS to
114            find path between nodes
115            Accepts graph, source node, sink node,
116            returns list of nodes, which is the path from src
117            to sink
118        """
119
120        # create queue, discovered list, previous
121        # list
122        queue, discovered, previous = Path.
123        build_bfs_structs(graph, source)
124
125        # recursive call
126        previous = Path.BFS(
127            graph=graph,
128            queue=queue,
129            discovered=discovered,
130            target=sink,
131            previous=previous,
132            neighbours=neighbours,
133        )
134
135        return Path._build_path(previous, sink)

136    @staticmethod
137    def _build_path(
138        previous: prev_map, sink: src.simplify.
139        graph_objects.Vertex
140    ) -> list[src.simplify.graph_objects.Vertex]:

```

```

136         """Given a mapping of previous nodes,
137         reconstructs a path to sink"""
138         path: list[src.simplify.graph_objects.
139             Vertex] = [sink]
140
141         while current := previous[path[0]]:
142             path.insert(0, current)
143
144         if path[0] == sink:
145             path = []
146
147     return path
148
149     @staticmethod
150     def BFS(
151         *,
152         graph: graphs.GenericDigraph,
153         queue: BFSQueue,
154         discovered: disc_map,
155         target: src.simplify.graph_objects.Vertex
156         | None,
157         previous: prev_map,
158         neighbours: Callable,
159         do_to_neighbour: Callable = void,
160         ) -> prev_map:
161
162         """BFS Search as part of finding the
163         shortest path through unweighted graph from src ->
164         target.
165
166         Target = None => walk through entire graph
167         , terminate at empty queue
168
169         Returns 'previous' list so that path can be
170         rebuilt.
171
172         Can pass in a function `do_to_neighbour` to
173         do to all nodes during the BFS
174
175         neighbour function need to return edge
176         """
177
178         # breakpoint: f"q: {queue}\n discovered: {str_map(discovered)}\nprev: {str_map(prev)}"
179
180         # will only happen if no path to node

```

```

168     if queue.is_empty():
169         return previous
170
171     else:
172         # discover next node in queue
173         current = queue.dequeue()
174         discovered[current] = True
175
176         # check we haven't been fed a
177         # standalone node (i.e. no forward or backwards links
178         )
179         if not graph.connected(current):
180             if not queue:
181                 raise SearchError("Cannot
182                 traverse a non connected node", current)
183
184             # if discovered target node return prev
185             if current == target:
186                 return previous
187
188             # otherwise, continue on
189             # enqueue neighbours, keep track of
190             # whose neighbours they are given not already
191             # discovered
192             # do passed in function to
193             # neighbouring nodes
194             for neighbour in neighbours(current
195             ):
196                 if not discovered[neighbour.
197                 node]:
198                     previous[neighbour.node] =
199                     current
200                     queue.enqueue(neighbour.
201                     node)
202
203                     do_to_neighbour(current,
204                     neighbour.node)
205
206                     # recursive call on new state
207                     return Path.BFS(

```

```
198                 graph=graph,
199                 queue=queue,
200                 discovered=discovered,
201                 target=target,
202                 previous=previous,
203                 neighbours=neighbours,
204                 do_to_neighbour=do_to_neighbour
205             )
206
```



```
1 # coding=utf-8
2
3 """
4 Set up graph object to be used in condensing debt
5 settling
6 """
7
8
9 import copy
10
11 from src.simplify.graph_objects import Vertex, Edge
12
13
14 class GraphError(Exception):
15     ...
16
17
18 class Default:
19     def __gt__(self, other):
20         return True
21
22
23 class GenericDigraph:
24     def __init__(self, vertices: list[Vertex]) ->
25         None:
26         """
27             Sets up a graph given a list of vertices
28         """
29         # initialise with values being empty
30         # build dict checking each type as we go
31
32         self.graph: dict[Vertex, list[Edge]] = {
33             vertex: [] if self.sanitize(vertex) else
34         None # type: ignore
35             for vertex in vertices
36         }
37
38         self._backwards_graph: dict[Vertex, list[
39             Edge]] = copy.deepcopy(self.graph)
```

```

38     def __str__(self):
39         """Pretty print graph"""
40         out = ""
41         for node, adj_list in self.graph.items():
42             pretty_nodes = ""
43             for edge in adj_list:
44                 pretty_nodes += f"{str(edge).upper()}"
45             out += f"{str(node).upper()} -> {"
46             pretty_nodes}\n"
47
48     return out
49
50     def __len__(self):
51         return len(self.graph)
52
53     def __getitem__(self, item):
54         return self.graph[item]
55
56     def __eq__(self, other):
57         return self.graph == other
58
59     def to_dot(self, *, n: int = 0, title="graph"):
60         """prints dot representation of graph"""
61         dot_source = ""
62         for src, adj_list in self.graph.items():
63             for edge in adj_list:
64                 dot_source += f"{str(src)} -> {str(edge.node)}\n"
65                 if not adj_list:
66                     dot_source += f"\n{src}\n"
67
68         dot = graphviz.Source(f"digraph {{ {dot_source} }}")
69         dot.format = "svg"
70         dot.render(f"./graph_renders/{title}{n}")
71
72     def nodes(self) -> list[Vertex]:
73         """Returns nodes in the graph"""
74         return list(self.graph.keys())

```

```

75     @staticmethod
76     def edge_from_nodes(node: Vertex, list_: list[Edge]) -> Edge:
77         """Checks if node is in a list of edges,
78         will return relevant edge if found
79         usage:"""
80         for edge in list_:
81             if edge.node == node:
82                 return edge
83             else:
84                 continue
85
86         raise GraphError("Node not in list")
87
88     @staticmethod
89     def nodes_from_edges(edges: list[Edge]) -> list[Vertex]:
90         nodes = []
91         for edge in edges:
92             nodes.append(edge.node)
93         return nodes
94
95     @staticmethod
96     def sanitize(v: Vertex, *args) -> bool:
97         """Raises GraphGenError if args are not
98         Vertex"""
99         if args is not None:
100             tests = [v, *args]
101         else:
102             tests = [v]
103             for test in tests:
104                 if type(test) is not Vertex:
105                     raise GraphError(f"{test} if of
106 type {type(test)} not Vertex ")
107
108     def is_node(self, v: Vertex) -> bool:
109         return v in self.graph
110
111     def add_node(self, v: Vertex) -> None:

```

```

111         self.graph[v] = []
112         self._backwards_graph[v] = []
113
114     def pop_node(self, v: Vertex) -> dict[Vertex,
115                                         list[Edge]]:
115         """Pops node, returns key/value pair of
116         node and previous connections"""
116         # look at _backwards_graph to find
117         # associations
117         for edge in self._backwards_graph[v]:
118             # removing B from A and C; A's pass
119             pointing_node_neighbours = self.
120             _backwards_graph[
121                 edge.node
121             ] # [Edge(A), Edge(C)]
122             pointing_edge = self.edge_from_nodes(v
122             , self.graph[edge.node])
123             self.graph[edge.node].remove(
123             pointing_edge) # type: ignore
124
125         return {v: self.graph.pop(v)}
126
127     def is_edge(self, s: Vertex, t: Vertex) -> int:
128         """Checks for an edge between nodes (
128         directional: s->t !=> t->s)"""
129         try:
130             self.edge_from_nodes(t, self.graph[s])
131             return 1
132         except GraphError:
133             return 0
134
135     def pop_edge(self, s: Vertex, t: Vertex) ->
135         Edge:
136         self.sanitize(s, t)
137         edge = self.edge_from_nodes(t, self.graph[s
137         ])
138
139         if edge is None:
140             raise GraphError("Cannot pop edge that
140             doesnt exist")
141

```

```
142         self.graph[s].remove(edge)
143     return edge
144
145     def neighbours(self, node: Vertex) -> list[Edge]:
146         self.sanitize(node)
147         return self[node]
148
149     def connected(self, node: Vertex) -> bool:
150         """Returns true if node has connections to
graph"""
151         forwards = self[node]
152         backwards = self._backwards_graph[node]
153         return True if forwards or backwards else
154             False
155
156 class Digraph(GenericDigraph):
157     def add_edge(self, s: Vertex, *args: Vertex
158 ) -> None:
159         self.sanitize(s, *args)
160         for target in args:
161             self.graph[s].append(Edge(target))
162             self._backwards_graph[target].append(
163                 Edge(s))
```



```
1 # coding=utf-8
2 from dataclasses import dataclass
3
4 from src.simplify.base_graph import GenericDigraph,
5     GraphError
6 from src.simplify.graph_objects import Vertex
7
8 """
9     Flow Graph"""
10
11
12
13
14 class FlowEdgeError(Exception):
15     ...
16
17
18 @dataclass(init=False, repr=False, eq=False)
19 class FlowEdge:
20     def __init__(self, src: Vertex, node: Vertex,
21                  capacity: int, flow: int = 0):
22         self.src: Vertex = src
23         self.node: Vertex = node # where is edge
24         pointing
25         self.capacity: int = capacity # non -ve;
26         self.flow: int = flow # always initialised
27         to 0
28         self.residual: bool = not capacity
29
30     def __str__(self):
31         return (
32             f"{self.node} [{self.flow}/{self.
33             capacity}], "
34             if not self.residual
35             else f"R: {self.node} [{self.flow}/{self
36             .capacity}],"
37         )
38
39     def __repr__(self):
40         return self.__str__()
```

```

36
37     def __eq__(self, other):
38         # TODO: write this properly
39         return str(self) == str(other)
40
41     def __lt__(self, other):
42         return self.capacity < other.capacity
43
44     def to_dot(self):
45         base = f'[label=" {self.flow}/{self.
capacity} "]'
46         return base[:-1] + ", color=red]" if self.
residual else base
47
48     def unused_capacity(self):
49         """Returns unused capacity of the edge"""
50         return self.capacity - self.flow
51
52     def push_flow(self, flow: int):
53         """Pushes flow down an edge; raises error if
too much"""
54         current = self.flow
55         if (new := flow + current) > self.capacity:
56             print(new, self.unused_capacity())
57             # raise error
58             raise FlowEdgeError(
59                 f" Tried to add {flow} units of flow"
60                 f" to an edge with {self.
unused_capacity()} units of flow available, "
61                 f"totalling {new}/{self.capacity}
units"
62             )
63         else:
64             self.flow = new
65
66     def adjust_edge(self):
67         """Changes edge values so that capacity =
unused_capacity, flow = 0"""
68         if self.residual:
69             # reset residual edge
70             self.flow = 0

```

```

71
72         else:
73             # make capacity = to what was unused
74             # capacity, unless unused capacity 0;
75             # if unused capacity 0, raise
76             # EdgeCapacityZero
77             if new := self.unused_capacity():
78                 self.capacity = new
79                 self.flow = 0
80             else:
81                 raise EdgeCapacityZero(self, new)
82
83
84
85
86
87     # map to keep track of people's net debts
88     # ; +ve if owes group, -ve if owed by group
89     self.net_debt: dict[Vertex, int] = {node: 0
90     for node in vertices}
91
92     def __bool__(self):
93         """Returns true if not empty"""
94         return not ({node: [] for node in self.
95         nodes()} == self.graph)
96
97     @staticmethod
98     def edge_from_nodes(node: Vertex, list_: list[
99         FlowEdge]) -> FlowEdge: # type: ignore
100        """gets edge from a list of edges (i.e. an
101        adjacency list) by node;
102        doesn't discriminate against residual edges
103        """
104
105        return GenericDigraph.edge_from_nodes(node
106        , list_) # type: ignore

```

```

103
104     def get_edge(self, src: Vertex, dest: Vertex
105         ) -> FlowEdge:
106         """Gets edge in graph between two nodes,
107         residual edges included"""
108
109         for node in [src, dest]:
110             self.sanitize(node)
111
112         return GenericDigraph.edge_from_nodes(dest
113             , self[src]) # type: ignore
114
115     def is_edge(self, s: Vertex, t: Vertex, *,
116     residual=False) -> bool:
117         """Checks for edge in a graph; residual =
118         True allows broadening to include residual edges"""
119
120         try:
121             edge = self.edge_from_nodes(t, self[s])
122             if residual:
123                 return True
124             else:
125                 return False if edge.residual else
126             True
127
128         except GraphError:
129             return False
130
131     def add_edge(self, src: Vertex, *edges: tuple[
132         Vertex, int], update_debt=True):
133         """Add a FlowEdge to graph, and also add a
134         residual edge"""
135
136         for dest, capacity in edges:
137             # make sure nodes in graph
138             self.sanitize(src, dest)
139
140             # no existing edge between two nodes
141             if not (self.is_edge(src, dest) or self
142                 .is_edge(dest, src)):
143                 # add normal edge
144                 self[src].append(FlowEdge(src, dest
145                     , capacity))
146
147                 # add residual edge

```

```

134                     self[dest].append(FlowEdge(dest,
135                         src, 0))
136                     self[src].sort(reverse=True)
137
138             # edge going in direction of edge being
139             added
140             elif self.is_edge(src, dest):
141                 self.get_edge(src, dest).capacity
142                 += capacity
143
144             elif self.is_edge(dest, src, residual=
145             False):
146                 new_cap = self.get_edge(dest, src).
147                 capacity - capacity
148                 # if new capacity is 0 pop edge
149
150                 if new_cap > 0:
151                     self.get_edge(dest, src).
152                     capacity = new_cap
153
154                     if new_cap < 0:
155                         self.pop_edge(dest, src)
156                         self.add_edge(src, (dest,
157                             new_cap * -1))
158
159                     if new_cap == 0:
160                         self.pop_edge(dest, src)
161
162
163             if update_debt:
164                 # handle net_debt;
165                 self.net_debt[src] += capacity
166                 self.net_debt[dest] -= capacity
167
168         def pop_edge(self, src: Vertex, dest: Vertex
169             , *, update_debt=False):
170             """removes REAL edges, and deletes residual
171             counterpart"""
172
173             # normal
174             fwd_edge = self.edge_from_nodes(dest, self[
175                 src])

```

```

165         self[src].remove(fwd_edge)
166         # residual
167         res = self.edge_from_nodes(src, self[dest])
168         if res.node == src:
169             # assert self[dest][0] == self[dest][1]
170             self[dest].remove(res)
171
172     else:
173         raise ValueError(
174             f" Tried to pop residual edge;
intended {dest} -> {src}, got {dest} -> {res.node}"
175             )
176
177     if update_debt:
178         # handle net_debt;
179         self.net_debt[src] -= fwd_edge.capacity
180         self.net_debt[dest] += fwd_edge.
capacity
181
182     def flow_neighbours(self, node: Vertex):
183         """returns neighbours of a node including
those related by residual edge
184         edges are valid iff they have unused
capacity"""
185
186         # build list of edges from adj list with
unused capacity
187         return [edge for edge in self[node] if edge
.unused_capacity()]
188
189     def adjust_edges(self):
190         """Run edge adjust on each edge, if new
capacity = 0 and residual = false,
191         delete edge + residual counterpart"""
192
193         edge: FlowEdge
194
195         for n, node in enumerate(self.nodes()):
196             for edge in self[node]:
197                 try:
198                     edge.adjust_edge()

```

```
199             except EdgeCapacityZero:
200                 # edge no longer has a place in
201                     my graph
202                         # delete edge + residual; will
203                             condition only ever raised on fwd edges
204                             self.pop_edge(node, edge.node)
205
206     # def nodes(self):
207         #     """Returns nodes in order of incoming
208             edges (smallest first)"""
209         #     def incoming_func(node):
210             #         return len([edge for edge in self[
211                 node] if edge.residual])
212             #     incoming = [(node, incoming_func(node))
213                 for node in self.graph.keys()]
214                 incoming.sort(key=lambda row: row[1],
215 reverse=True)
216             #     return [node for node, _ in incoming]
```



```
1 # coding=utf-8
2
3 from dataclasses import dataclass
4
5
6 """All objects to be used in various graphs"""
7
8
9 class FlowError(Exception):
10     ...
11
12
13 @dataclass
14 class Vertex:
15     """Representation of a vertex; carries data and
16     ID"""
17     ID: int # IDs should be unique
18     label: str = "" # optional label
19
20     def __key(self) -> tuple:
21         """Returns immutable repr of object for
22         hashing"""
23         return self.ID, self.label
24
25     def __str__(self):
26         return self.label if self.label else str(
27             self.ID)
28
29     def __hash__(self):
30         """for adding to lists / dicts"""
31         return hash(bytes(f"{{self.__key()}}".encode("utf8")))
32
33 @dataclass
34 class Edge:
35     node: Vertex
36
37     def __str__(self):
38         return str(self.node)
```

```

38
39     def to_dot(self):
40         """str repr for dot"""
41         return ""
42
43
44 @dataclass
45 class WeightedEdge(Edge):
46     weight: int
47
48     def __str__(self):
49         return f"{self.node} [{self.weight}], "
50
51     def to_dot(self):
52         return f"[label={self.weight}]"
53
54
55 @dataclass
56 class FlowEdge(Edge):
57     capacity: int = 0
58     flow: int = 0
59     residual: bool = False # class as residual edge
60     if capacity is 0
61
62     def __str__(self):
63         return (
64             f"{self.node} [{self.flow}/{self.
65             capacity}], " if not self.residual else ""
66         )
67
68     def to_dot(self):
69         base = f'[label="{self.flow}/{self.capacity}"
70                 ]'
71         return base[:-1] + ", color=red]" if self.
72             residual else base
73
74     def unused_capacity(self) -> int:
75         return self.capacity - self.flow
76
77     def push_flow(self, flow):
78         if self.flow + flow > self.capacity or type(

```

```
74 flow) is not int:  
75         raise FlowError(  
76             f"Cannot send {flow} units down  
77             path of capacity {self.capacity}"  
78         )  
79         self.flow += flow  
80  
81     def __str__(self):  
82         return f"Flow object with flow {self.flow} and capacity {self.capacity}"
```



```
1 # coding=utf-8
2
3 import copy
4
5 from src.simplify import path as path
6 from src.simplify.flow_graph import FlowGraph,
    FlowEdge
7 from src.simplify.graph_objects import Vertex
8
9
10 class SettleError(Exception):
11     ...
12
13
14 class NoOptimisations(Exception):
15     """No optimisations to graph; already in
    simplest form"""
16
17
18 class MaxFlow:
19     @staticmethod
20     def edmonds_karp(graph: FlowGraph, src: Vertex,
    sink: Vertex) -> int:
21
22         max_flow = 0
23
24         while aug_path := MaxFlow.augmenting_path(
    graph, src, sink):
25             bottleneck = MaxFlow.bottleneck(graph,
    aug_path)
26             max_flow += bottleneck
27
28             MaxFlow.augment_flow(graph, aug_path,
    bottleneck)
29             # graph.to_dot()
30
31         return max_flow
32
33     @staticmethod
34     def augmenting_path(graph: FlowGraph, src:
    Vertex, sink: Vertex) -> list[Vertex]:
```

```

35         """find the shortest path from src -> sink
36         using BFS"""
37         return path.Path.shortest_path(
38             graph, src, sink, neighbours=graph.
39             flow_neighbours
40         )
41     @staticmethod
42     def bottleneck(graph: FlowGraph, node_path: list
43 [Vertex]) -> int:
44         """Returns bottleneck of a path"""
45         aug_path = MaxFlow.nodes_to_path(graph,
46         node_path)
47         remaining = [edge.unused_capacity() for edge
48           in aug_path]
49         return min(remaining)
50
51     @staticmethod
52     def augment_flow(graph: FlowGraph, node_path:
53 list[Vertex], flow: int) -> None:
54         """Adds flow to normal edges of path,
55         subtracts from residual
56         deals with pushing to residual and hence
57         subtracting from normal"""
58         # normal edges
59         aug_path = MaxFlow.nodes_to_path(graph,
60         node_path)
61         for edge in aug_path:
62             edge.push_flow(flow)
63
64         # flip node path to give residual
65         node_path.reverse()
66         # get edges from reversed path
67         res_path = MaxFlow.nodes_to_path(graph,
68         node_path)
69
70         # push flow down res path
71         for edge in res_path:
72             edge.push_flow(flow * -1)
73
74     @staticmethod

```

```

66     def nodes_to_path(graph: FlowGraph, nodes: list[Vertex]) -> list[FlowEdge]:
67         graph.sanitize(*nodes)
68
69         edges: list[FlowEdge] = []
70         for src, neighbour in zip(nodes, nodes[1:]):
71             edges.append(graph.get_edge(src, neighbour))
72
73     return edges
74
75
76 class Simplify:
77     @staticmethod
78     def simplify_debt(debt: FlowGraph) -> FlowGraph:
79         """
80         for edge(u, v) in graph:
81             if new := maxflow(u, v):
82                 clean.add_edge(u, (v, new))
83                 messy.adjust_edges()
84         """
85
86         clean = FlowGraph(debt.nodes())
87
88         d_cache = copy.deepcopy(debt)
89
90         # iterate through edges in graph:
91         while not debt:
92             edge: FlowEdge # type: ignore
93             for (node, adj_list) in debt.graph.items():
94
95                 for edge in adj_list: # type:
96                     ignore
97                     if not edge.residual:
98                         if flow := MaxFlow.edmonds_karp(debt, node, edge.node):
99                             clean.add_edge(node, (edge.node, flow))

```

```
99                      debt.adjust_edges()
100
101         if clean == d_cache:
102             clean.to_dot(n=1)
103             raise NoOptimisations
104
105         if d_cache.net_debt != clean.net_debt:
106             raise SettleError("Settling failed;
debt was skewed")
107
108         return clean
109
```

```

1 # coding=utf-8
2
3 from src.simplify.base_graph import GenericDigraph,
4     GraphError
5
6
7 class WeightedDigraph(GenericDigraph):
8     def add_edge(self, source: Vertex, *edges: tuple[Vertex, int]) -> None:
9
10         # sanitize source
11         self.sanitize(source)
12         for node, weight in edges:
13             if self.is_edge(source, node):
14                 existing: WeightedEdge
15                 existing = self.edge_from_nodes(node
16 , self[source]) # type: ignore
17                 existing.weight += weight
18             else:
19                 self.sanitize(node)
20                 self.graph[source].append(
21                     WeightedEdge(node, weight))
22                     self._backwards_graph[node].append(
23                     WeightedEdge(source, weight * -1))
24
25         def flow_through(self, node: Vertex) -> int:
26             """Returns sum of weights out of node
27             +ve => net flow out of node
28             -ve => net flow into node"""
29             flow = 0
30             edge: WeightedEdge
31
32             for graph in [self.graph, self.
33             _backwards_graph]:
34                 for edge in graph[node]: # type: ignore
35                     flow += edge.weight
36
37             # return -ve, as backwards edges have -ve
38             # value and forwards have +ve

```

```
34         return flow
35
36     def net_debts(self) -> dict[Vertex, int]:
37         """Returns a map of everyone with net money
38             owed (-ve if they need to pay)"""
39         return {node: self.flow_through(node) for
40             node in self.nodes()}
41
42     def is_edge(self, s: Vertex, t: Vertex) -> int:
43         try:
44             edge: WeightedEdge
45             edge = self.edge_from_nodes(t, self.
46                 graph[s]) # type: ignore
47             return edge.weight
48         except GraphError:
49             return 0
```

```
1 # coding=utf-8
2
3 import csv
4 import os
5 from dataclasses import dataclass, field
6
7 import src.simplify.flow_algorithms
8 import src.simplify.flow_graph as flow
9 from src.crypto import keys
10 from src.simplify.flow_algorithms import Simplify,
11     SettleError
12 from src.simplify.graph_objects import Vertex
13 from src.transactions.transaction import Transaction
14     , VerificationError
15
16
17
18
19 class LedgerBuildError(Exception):
20     """"Error building ledger"""
21
22
23 @dataclass
24 class Ledger:
25     """"Multiple transactions contained to one group
26     (assumed from building);
27     built from a stream of transaction objects"""
28
29     # ledger, big list of transactions;
30     # TODO: maybe make ledger generator
31     ledger: list[Transaction] = field(
32         default_factory=lambda: [])
33     nodes: list[Vertex] = field(default_factory=
34         lambda: [])
35     key_map: dict[int, keys.RSAPublicKey] = field(
36         default_factory=lambda: {})

37
38     def __bool__():
39         """"False if ledger empty"""

40
```

```

36         return not not self.ledger
37
38     def __eq__(self, other):
39         return self.ledger == other.ledger
40
41     def append(self, transaction: Transaction) ->
42         list[Transaction]:
42         """Nice syntax for adding transactions to
43         ledger"""
43
44         if type(transaction) is not Transaction:
45             raise LedgerBuildError(
46                 f"cannot append type {transaction}
47                 to ledger; must be transaction"
48             )
49         else:
50             self.ledger.append(transaction)
50             self.key_map[transaction.src] =
51                 transaction.src_pub
51             self.key_map[transaction.dest] =
52                 transaction.dest_pub
52
53         return self.ledger
54
55     def _verify_transactions(self) -> None:
56         """Verifies the keys of all the transactions
56         in the group.
57         Raises error if a faulty transaction is
57         found"""
58
59         for trn in self.ledger:
60             trn.verify()
61
62     def _as_flow(self) -> flow.FlowGraph:
63         """Returns ledger as a flow graph"""
64         # Extract IDs involved -> nodes
65         nodes: set[Vertex] = set()
66         for trn in self.ledger:
67             nodes.add(Vertex(trn.src))
68             nodes.add(Vertex(trn.dest))
69

```

```

70         self.nodes = list(nodes)
71         self.nodes.sort(key=lambda node: node.ID)
72
73         # build flow graph with nodes
74         as_flow = flow.FlowGraph(self.nodes)
75
76         # verify transaction, add to graph
77         for trn in self.ledger:
78             trn.verify()
79             as_flow.add_edge(Vertex(trn.src), (
80                 Vertex(trn.dest), trn.amount))
81
82         return as_flow
83
83     def _flow_to_transactions(self, fg: flow.
84         FlowGraph) -> list[Transaction]:
85         """For each edge, make a transaction"""
86
86         new_trns: list[Transaction] = []
87
88         edge: flow.FlowEdge
89
90         # go through every outgoing transaction by
91         # person
92         for person, adj_list in fg.graph.items():
93             for edge in adj_list: # type: ignore
94
94                 # don't add residual edges to
95                 # ledger
95                 if edge.residual:
96                     continue
97
98                 else:
99                     # generate UNSIGNED
100                     # transactions
100                     # TODO: let db handle id; keep
101                     # it initialised to 0
101                     edge: flow.FlowEdge # type:
102                     # ignore
102                     trn = Transaction(
103                         edge.src.ID,

```

```

104                     edge.node.ID,
105                     edge.capacity,
106                     self.key_map[edge.src.ID],
107                     self.key_map[edge.node.ID],
108                     )
109
110                     new_trns.append(trn)
111
112     return new_trns
113
114     def simplify_ledger(self):
115         # build ledger as a flow graph
116         fg = self._as_flow()
117         fg.to_dot(title="pre_settle")
118         try:
119             simplified_fg = Simplify.simplify_debt(
120                 fg)
121
122             # settle, update ledger
123             simplified_fg.to_dot(title="settled")
124             self.ledger = self.
125             _flow_to_transactions(simplified_fg)
126
127             except SettleError:
128                 # no changes made to graph, keep ledger
129                 # as is, with sigs.
130                 # print(
131                 #     "Graph already at few
132                 # transactions per person; no optimisations found"
133                 # )
134                 raise SettleError('Failed to settle
135                 graph... aborted')
136
137             except src.simplify.flow_algorithms.
138             NoOptimisations:
139                 raise NoFurtherSimplifications(
140                     "Graph already at few transactions
141                     per person; no optimisations found"
142                     )
143
144             except VerificationError as ve:

```

```

138                 # let verification error propagate up
139                 raise ve
140
141
142 class LedgerLoader:
143     @staticmethod
144     def load_from_csv(path: str) -> list[Ledger]:
145         """Load from a csv, in transaction format
146
147         print("loading from csv")
148
149         def get_field(str_: str) -> int:
150             return header.index(str_)
151
152         def build_trn() -> tuple[
153             int, int, int, keys.RSAPublicKey, keys.
154             RSAPublicKey, int
155         ]:
156
157             ldr = keys.RSAKeyLoaderFromNumbers()
158             ldr.load(n=int(row[get_field("src_n"]
159             ])), e=int(row[get_field("src_e")])) # type:
160             ignore
161             src_pub: keys.RSAPublicKey = keys.
162             RSAPublicKey(ldr)
163
164             ldr2 = keys.RSAKeyLoaderFromNumbers()
165             ldr2.load(n=int(row[get_field("dest_n"]
166             ])), e=int(row[get_field("dest_e")])) # type:
167             ignore
168             dest_pub: keys.RSAPublicKey = keys.
169             RSAPublicKey(ldr2)
170
171             print(src_pub, dest_pub, "----", sep="\n"
172             )
173
174             return (
175                 int(row[get_field("src")]),
176                 int(row[get_field("dest")]),
177                 int(row[get_field("amount")]),
178             )

```

```
170                 src_pub,
171                 dest_pub,
172                 int(row[get_field("ID")]),
173             )
174
175         if not os.path.exists(path):
176             raise FileNotFoundError(
177                 f"File not found at current path: \
178                 {os.getcwd()};\nsearched for {path}"
179             )
180
180     transactions: list[list[Transaction]] = []
181
182     # generate transaction objects, store as
183     # list of groups of transactions
184     with open(path) as csvfile:
185         transaction_reader = csv.reader(csvfile
186         , delimiter=",")
187         for row in transaction_reader:
188             # use header to build index of
189             # where things are
190             if row[0] == "ID":
191                 header: list[str] = row
192                 continue
193
194
195             try:
196                 transactions[int(row[get_field(
197                     "group")])].append(trn)
198             except IndexError:
199                 # make position at group if not
200                 # made yet (assuming consecutive 0 indexed group
201                 # numbers
202                 transactions.append([trn])
203
204     ledgers: list[Ledger] = []
```

```
203
204     for group in transactions:
205         ledger = Ledger()
206         for trn in group:
207             ledger.append(trn)
208         ledgers.append(ledger)
209
210     return ledgers
211
```



```
1 # coding=utf-8
2
3 """
4 Handles transaction object
5 """
6
7 import datetime
8
9 # coding=utf-8
10 import sys
11 from abc import ABC, abstractmethod
12 from dataclasses import dataclass, field
13
14 from src.crypto import keys, hashes, rsa
15
16
17 class TransactionError(Exception):
18     ...
19
20
21 class VerificationError(Exception):
22     ...
23
24
25 class Signable(ABC):
26     """Base class for objects that can be signed;
27     needs a hash implementation, cannot be added into
28     ABC for reasons"""
29
30     @abstractmethod
31     def sign(self, key: keys.RSAPrivateKey, *,
32             origin: str) -> None:
33         ...
34
35     @abstractmethod
36     def hash(self) -> bytes:
37         ...
38
39
40     @dataclass
41     class Transaction(Signable):
```

```

39     src: int
40     dest: int
41     amount: int
42     src_pub: keys.RSAPublicKey
43     dest_pub: keys.RSAPublicKey
44     ID: int = 0
45     reference: str = ""
46     time: datetime.datetime = datetime.datetime.now()
47     signatures: dict[int, bytes] = field(
48         default_factory=lambda: {})
49     group: int = 0
50
51     def hash(self) -> bytes:
52         # standard way to produce hash using SHA256
53         # interface from crypto lib
54         return (
55             hashes.Hasher(
56                 f"{self.src, self.dest, self.amount,
57                  self.reference, self.time}".encode(
58                     "utf8", sys.byteorder
59                 )
60             )
61             .digest()
62             .h
63         )
64
65         # note: IMPORTANT: CANNOT USE DUUNDER HASH AS
66         # PYTHON DOES WEIRD HASH COMPRESSION
67         def __hash__(self):
68             raise NotImplementedError("__hash__ cannot
69             be used, used .hash() method")
70
71         def sign(self, key: keys.RSAPrivateKey, *,
72             origin: str) -> None:
73             def overwrite_err(where: str) -> None:
74                 """Helper to raise overwrite error"""
75                 raise TransactionError(f"Cannot
76                 overwrite signature of {where}")
77
78             def generate_signature(obj: Transaction) ->

```

```

71 bytes:
72             """Helper to generate a signature of
73             the object"""
74             sig = rsa.RSA.sign(obj.hash(), key)
75             return sig
76
77             # raise error if not given a priv key
78             if (keytype := type(key)) is not keys.
    RSAPrivateKey:
79                 raise TransactionError(f"Was given a {
    keytype}, not an RSAPrivateKey")
80
81             # initialise sigs dict if it doesn't
82             # already exist
83             if not self.signatures:
84                 self.signatures = {self.src: b"",
    dest: b""}
85             else:
86                 # check that, if only one signature
87                 # exists, the other is created as null
88                 try:
89                     self.signatures[self.src]
90                 except KeyError:
91                     self.signatures[self.src] = b""
92
93                 try:
94                     self.signatures[self.dest]
95                 except KeyError:
96                     self.signatures[self.dest] = b""
97
98                 # accept origin as src or dest;
99                 # check for overwrite, raise error if case
100                # append with key of ID to signatures dict
101                if origin == "src":
102                    if not self.signatures[self.src]: #
    check for overwrites
103                        self.signatures[self.src] =
    generate_signature(self)
104                    else:
105                        overwrite_err("src") # handle
    overwrite err

```

```

103
104         elif origin == "dest":
105             if not self.signatures[self.dest]:
106                 self.signatures[self.dest] =
107                     generate_signature(self)
108             else:
109                 overwrite_err("dest")
110
111         else:
112             # if parameter wasn't src or dest,
113             raise error
114             raise ValueError(f"{origin!r} not a
115 valid parameter; use 'src' or 'dest'")
116
117     def verify(self) -> None:
118         """Raise verification error if invalid sig
119         """
120
121         # ensure that transaction has its two
122         signatures:
123         try:
124             for sig in [self.src, self.dest]:
125                 _ = self.signatures[sig]
126         except KeyError:
127             raise VerificationError(f"Has not been
128 signed by USRID {sig}")
129
130         # build list of keys to verif
131         usr_keys = [self.src_pub, self.dest_pub]
132
133         if not usr_keys:
134             raise VerificationError("No valid keys
135 were passed in")
136
137         hash_from_obj: bytes = self.hash()
138
139         # note: if signature is a hex string
140         convert to bytes
141         for s_key, val in self.signatures.items():
142             if type(val) is str:
143                 self.signatures[s_key] = int(val,

```

```
135 16).to_bytes(256, sys.byteorder)
136
137     for key, origin in zip(usr_keys, [self.src
138         , self.dest]):
139         hash_from_sig = rsa.RSA.inv_sig(self.
140             signatures[origin], key)
141         if hash_from_sig != hash_from_obj:
142             raise VerificationError(f"self.src
143 } sig invalid")
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1390
1391
1392
1393
1394
1394
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2317
2318
2319
2320
232
```

Testing

Re-listed Low Level Requirements

The list of requirements low level requirements, identical to that in the Analysis section.

RSA Implementation (A)

1. A reliable interface to a hashing module
2. RSA Key Handling:
 1. Be able to load RSA public/private keys in PEM format from files / STDIN
 2. Be able to validate the format of these keys
 3. Be able to parse these keys extracting all necessary numbers for RSA decryption
3. Signing/Verification
 1. Have a valid RSA encryption scheme (encryption with public key)
 2. Have a valid RSA decryption scheme (decryption with private key)
 3. Have a valid RSA signing (sig) scheme (signing with private key)
 4. Have a valid RSA signature verification (verif) scheme (verify with public key)
4. Object Signing
 1. Algorithm to convert an object to a hash in a reproducible way, minimising the chance of hash collisions
 2. Ability to sign a class of object with RSA sig scheme
 3. Ability to verify a signed object with RSA verif scheme, raising an error if signature is invalid

Debt Simplification (B)

1. A reliable digraph structure, with operations to
 1. Get the nodes in the graph
 2. Check if an edge exists between two nodes
 3. Nodes can be added
 4. Nodes can be removed
 5. Edges can be added
 6. Edges can be removed
 7. Neighbours of a node should be easily accessed (neighbours for the purposes of a breadth first search)
2. A reliable flow graph structure
 1. All the operations listed in B.1.1

2. Adding an edge should have different functionality: edge should be able to be added with a capacity, and edges should have a notion of flow and unused capacity
3. Be able to return neighbours of nodes in the residual graph (i.e. edges, including residual edges, that have unused capacity)
4. A way to get the bottleneck value of a path, given a path of nodes

3. A reliable recursive BFS that works on flow graphs

4. Implementation of Edmonds-Karp
 1. Way to find the shortest augmenting path between two nodes
 2. Way to find bottleneck value of a path
 3. Finding max flow along a flow graph from source node to sink node

5. Simplifying an entire graph using Edmonds Karp, using the method laid out in [Settling a graph using a Max Flow algorithm](#).

6. Be able to convert a list of valid transactions into a flow graph

7. Be able to convert a flow graph into a list of transactions, signed by the server

8. Be able to simplify a group of transactions, having each transaction individually verified before settling

Client / Server Structure (C)

1. The server should be accessible to the client via a REST API
2. The client should be relatively thin, only dealing with input from user and handling error 400 and 500 codes gracefully.
3. Client and Server should communicate over HTTP, using JSON as an information interchange format
4. The client should have a clear, easy to use command line interface

'Integrated' requirements for how the end system should behave (D)

1. Ensuring the validity of transactions
 1. If a transaction is tampered with in the database, it should be classed as unverified
 2. A user should not be able to sign an already signed transaction
 3. A user should not be able to sign a transaction where they are not one of the listed members
 4. A user should not be able to sign a transaction with a key that is not associated to their account
 5. A user should not be able to sign a transaction without entering their password correctly
 6. Every time a transaction is pulled from the database and sent to the user, it should be verified by the server using the RSA sig/verif scheme from section A

2. Ensuring that the debt simplification feature works

1. All transactions in the group being settled should be verified upon being pulled from the database
2. It should not be possible to simplify a group if there are unverified transactions in the group
3. If the transaction structure of the group does not change, the user should be notified
4. The simplification should accurately simplify a system of debts such that no one is owed / owes a different amount of money after simplification
5. The simplifying process should result in unverified transactions being produced, able to be signed by the user

3. Ancillary features

1. Users should be able to register for an account, providing name, email, password and a PEM formatted private key
2. Users should be able to create transactions where they are the party owing money; these transactions should be created as unsigned
3. Users should be able to create a group with a name and password
4. Users should be able to join a group by group ID
5. Users should be able to mark a transaction as settled; transactions should only be marked as settled when both parties involved mark the transaction as settled
6. Users should be able to see which groups they are a member of
7. Users should be able to see all of their open transactions
8. Users should be able to see all the open transactions in a group (whether they are part of the group)
9. Users should be able to see the public key information of any user on the system
10. Users should be able to see individual transactions by passing in a transaction ID

Database Architecture (E)

1. User information

1. User ID
2. Contact info
3. A hash of the user's password
4. Associated Groups
5. Public Key (provisions for one or more)

2. Transaction Information

1. Transaction ID
2. Payee
3. Recipient
4. Transaction reference
5. Amount (£)
6. Payee's signature
7. Recipient's signature
8. Whether transaction has been settled

3. Group information
 1. Group name
 2. Group password
 3. People in the group
 4. Transactions in the group

Unit Test Framework

To demonstrate the effectiveness and completeness of sections A & B, I will provide my unit testing framework. I approached implementation from a test-driven development perspective, and thus all of these tests were written before the code they run was implemented.

This has led to the creation of an extensive, robust framework of tests, which effectively shows the extent to which I have completed sections A and B of my project.

The test harness has ~80% coverage on the `transactions`, `simplify` and `crypto` modules, with the crypto module having >90% coverage. This makes it hard to argue that my solution has not been adequately tested.

Below is a report of my tests running generated by my IDE, as well as a table linking individual unit tests to requirements from sections A and B. An interactive version of the file is included in the project files.

The unit tests are provided after the Evaluation.

(Note - the slightly long time that these tests took to run can be attributed to the drawing of graphs. I used the `graphviz` library to dynamically generate pictures of graphs of the debt that my algorithm was simplifying. Some of these are included below.)

: 100 total, 100 passed

2.45 s

[Collapse](#) | [Expand](#)

test_crypto

 test_hashes

 TestHash

 test_hash (tests that hash object can validate hash looking numbers)

 [valid]

 [too short]

 [too long]

 [wrong type]

600 ms

11 ms

11 ms

9 ms

passed

passed

passed

passed

 test_hasher_fails (Checks that hasher objects when not bytes are passed into it)

passed 0 ms

 test_init (Checks that _hasher is initialised correctly i.e. no strange start values)

passed 0 ms

 test_update_digest (Ensures that a hash with a given value will digest the correct thing)

2 ms

 [before update]

passed

 [after update]

passed

 test_keys

 TestRSAKeyLoading

 test_file_loading (Tests that file is being loaded correctly assuming correct file)

passed 99 ms

 /home/tcassar/projects/settle/src

 0

 test_file_not_found

passed 7 ms

 /home/tcassar/projects/settle/src

 0

 test_parsing

 [pub_exp]

passed

 [priv_exp]

passed

 [mod]

passed

 test_public_key

101 ms

 [allowed access]

passed

 [deny access]

passed

 test_unloaded_key

passed 7 ms

 /home/tcassar/projects/settle/src

 0

 test_unparsed_key (Check accessing attributes)

passed 58 ms

 /home/tcassar/projects/settle/src

 0

 test_wrong_format (Checks that we can deal with files being the wrong format)

passed 45 ms

 /home/tcassar/projects/settle/src

 0

 unable to load Private Key

 140267057518400:error:0909006C:PEM routines:get_name:no start line:../crypto/pem/pem_lib.c:745:Expecting: ANY PRIVATE KEY

[Collapse](#) | [Expand](#)

test_sign_verify		212 ms
TestRSA		212 ms
test_RSA_sign (Checks for consistent creating / verifying of a 'signature')	passed	108 ms
test_encryption (Checks to see if process is reversible, and encrypted is different to how it started)		104 ms
[Catch Public Key]	passed	
[encrypted]	passed	
[Successful decryption]	passed	
test_settling		2.92 s
test_Path		9 ms
TestPath		9 ms
test_build_bfs_struct		3 ms
[with initial value]	passed	
[with initial value]	passed	
test_build_path (Given a prev_map, check we build the right path)	passed	0 ms
test_find_target	passed	1 ms
test_recursive_bfs (Check that BFS is finding paths along graph correctly)	passed	1 ms
test_shortest_path (Checks that we can in fact find shortest path along)		4 ms
[digraph]	passed	
[weighted_graph]	passed	
[flow]	passed	
test_flow		2.90 s
TestFlowEdge		4 ms
test_adjust_edge	passed	1 ms
test_push_flow (Checks that we update flow by required amount)		2 ms
[R: 1 [0/0],]	passed	
[1 [3/5],]	passed	
[exceed capacity]	passed	
test_unused_capacity (builds two edges, residual and non-residual)		1 ms
[R: 1 [-3/0],]	passed	
[1 [0/5],]	passed	
TestFlowGraph		522 ms
test_add_edge		370 ms
[negative test]	passed	
[added]	passed	
[Net debt (adding)]	passed	
[Net debt (removing)]	passed	

test_bool		passed	0 ms
{Vertex(ID=0, label='a'): [], Vertex(ID=1, label='b'): [], Vertex(ID=2, label='c'): [], Vertex(ID=3, label='d'): [], Vertex(ID=4, label='e'): []}			
test_flow_neighbours (Checks we get edges that have unused capacity, including residual)		passed	143 ms
DEBUG:graphviz._tools.os.makedirs('./graph_renders')			
DEBUG:graphviz.saving.write lines to './graph_renders/graph0'			
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph0']			
test_get_edge		passed	0 ms
test_is_edge			1 ms
[no edge]		passed	
[edge]		passed	
test_pop_edge			8 ms
[negative]		passed	
[removed edge]		passed	
[removed residual]		passed	
TestMaxFlow			514 ms
test_augment_flow			172 ms
[a -> b]		passed	
[b -> c]		passed	
[c -> d]		passed	
[d -> c]		passed	
[c -> b]		passed	
[b -> a]		passed	
test_augmenting_path		passed	0 ms
test_bottleneck		passed	154 ms
DEBUG:graphviz._tools.os.makedirs('./graph_renders')			
DEBUG:graphviz.saving.write lines to './graph_renders/graph0'			
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph0']			
test_edmonds_karp		passed	181 ms
DEBUG:graphviz._tools.os.makedirs('./graph_renders')			
DEBUG:graphviz.saving.write lines to './graph_renders/graph0'			
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph0']			
test_nodes_to_path		passed	3 ms
test_old_edmonds		passed	4 ms
TestSimplify			1.86 s
test_adjust_edges		passed	588 ms
DEBUG:graphviz._tools.os.makedirs('./graph_renders')			
DEBUG:graphviz.saving.write lines to './graph_renders/graph0'			
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph0']			
DEBUG:graphviz._tools.os.makedirs('./graph_renders')			
DEBUG:graphviz.saving.write lines to './graph_renders/graph1'			
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph1']			

```
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph1']
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/graph4'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph4']
```

test_mithun_simplify

passed 391 ms

```
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/graph0'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph0']
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/graph1'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph1']
```

test_netflow (Checking people owed same before and after)

passed 0 ms

test_simplest_graph (Shouldn't change)

passed 496 ms

```
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/graph0'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph0']
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/graph1'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph1']
```

test_simplify_debt

passed 389 ms

```
{Vertex(ID=0, label='d'): 15, Vertex(ID=1, label='m'): 0, Vertex(ID=2, label='t'): -15}
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/graph0'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph0']
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/graph1'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'graph1']
```

test_graph

6 ms

TestDigraph

5 ms

test_add_edge

passed 1 ms

test_add_node

passed 0 ms

test_init

1 ms

[init]

passed

[helpers]

passed

test_is_edge

2 ms

[is edge]

passed

[isn't edge]

passed

test_nodes

passed 0 ms

test_pop_edge

passed 0 ms

test_pop_node

passed 1 ms

```
U -> VW
V -> W
W ->
V -> W
W ->
```

TestWeightedDigraph

1 ms

[Collapse](#) | [Expand](#)

test_add_edge	passed	0 ms
test_add_existing_edge	passed	0 ms
test_flow_through		1 ms
[u]	passed	
[v]	passed	
[w]	passed	
test_transactions		3.93 s
test_ledger		3.44 s
TestLedger		3.44 s
test_add		234 ms
[Add]	passed	
[Catch non transaction]	passed	
test_as_flow		715 ms
[nodes]	passed	
[to flow graph]	passed	
test_flow_to_transactions		551 ms
[to transactions]	passed	
test_load_from_csv	passed	285 ms
loading from csv		
n=216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511:		
e=65537		
n=245435260532226859200890024313870710808087539791682893396507074141095827708342421730095392824267456827229880553467004005448679109437851439		
e=65537		

n=216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511:		
e=65537		
n=225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676		
e=65537		

n=245435260532226859200890024313870710808087539791682893396507074141095827708342421730095392824267456827229880553467004005448679109437851439		
e=65537		
n=225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676		
e=65537		

n=216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511:		
e=65537		
n=245435260532226859200890024313870710808087539791682893396507074141095827708342421730095392824267456827229880553467004005448679109437851439		
e=65537		

n=216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511:		
e=65537		
n=225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676		
e=65537		

n=245435260532226859200890024313870710808087539791682893396507074141095827708342421730095392824267456827229880553467004005448679109437851439		
e=65537		
n=225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676		

Crypto (A)	File	Test(s)
A1	test_crypto.test_hashes	*
A2	test_crypto.test_keys	
		test_file_not_found
		test_file_loading
		test_wrong_format
		test_parsing
		test_unparsed_key
		test_unloaded_key
		test_public_key
A3		test_sign_verify
		test_encryption
		test_encryption
		test_RSA_sign
		test_RSA_sign
A4	test_transactions.test_transaction	
		test_hash
		test_sign
		test_verify
Simplify (B)	test_settling	
B1	test_graph	
	B1.1	test_nodes
	B1.2	test_is_edge
	B1.3	test_add_node
	B1.4	test_pop_node
	B1.5	test_add_edge
	B1.6	test_pop_edge
	B1.7	
B2	test_flow.TestFlowGraph	
	B2.1	tests by the same name as above
	B2.2	test_add_edge
	B2.3	test_flow_neighbours
	B2.4	test_bottleneck
B3	test_Path	
	B3.1	All tests in file
	B3.2	All tests in file
B4	test_flow.TestMaxFlow	
	B4.1	test_augmenting_path
	B4.2	test_bottleneck
	B4.3	test_nodes_to_path test_augment_flow test_edmonds_karp
B5	test_flow.TestSimplify	
		All tests in TestCase
B6	test_transactions.test_ledger	
B7		test_as_flow
		test_flow_to_transactions
		test_verify_transactions
B8		test_simplify_ledger

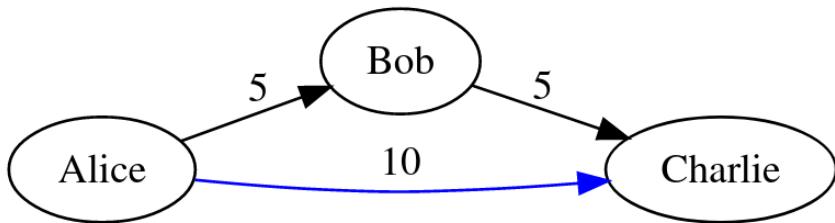
I appreciate this table may not show information about the individual tests. Every unit test I wrote has an informative docstring. Thus, refer to the code after the Evaluation for more clarification about the function of any unittest.

Proof of simplification algorithm functionality (B5)

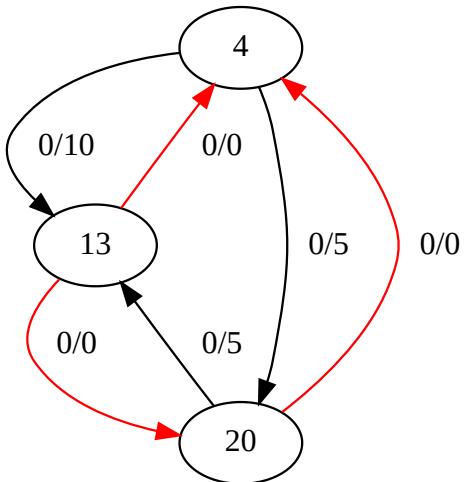
In the analysis section, I provided a hand-trace of the expected graph structures involved in settling a system of debts. Having done this hand tracing, I decided to use the same graph to test my algorithm.

Using the `graphviz` library I was able to generate before and after representations of the flow graphs used to simplify the system of debt it was fed. Here is my initial structure compared to the expected structure

Expected



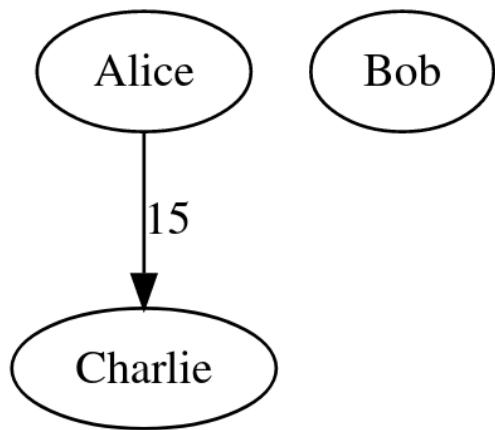
Generated



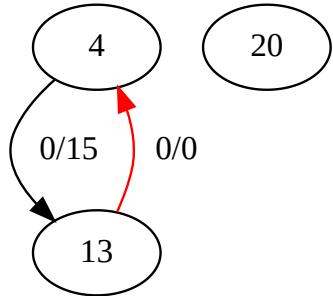
Since this is debug output, IDs are displayed instead of names and residual edges and flows are shown. Here, red edges can be ignored, and you can take the debt along the edge to equal the capacity of the edge.

Though the identifiers are different, the generated graph is in the form of the expected graph

Expected outcome



Generated outcome



It is clear from these diagrams that the settling process has worked as expected, matching up perfectly with the hand-traced data. The same results are shown in the CLI of the technical solution

The first call `settle verify -g 11` returns all unsettled transactions stored by the database, and checks whether their signatures are valid. In this case, all signatures are valid and thus the group can be simplified. Transactions of the form of the graphs above are found in the group.

The second call is to `settle simplify 11`. This indicates to the server that group 11 should be simplified. The group's password has been entered correctly and thus simplification can occur.

The simplification process will then verify each transaction once more, and build a flow graph representation of the system of debts. My algorithm based on Edmonds-Karp is then run, and a single new transaction is generated. Again, this new transaction is in the form of the graphs above. proving the effectiveness of the solution

```
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -g 11
----  

Transaction ID = 14  

Group: 11  

cassar.thomas.e@gmail.com -> mreymacia@gmail.com, £5.00  

simplify test  

at 2022-03-22 14:57:10.711745  

Verified: True  

----  

Transaction ID = 15  

Group: 11  

cassar.thomas.e@gmail.com -> kezza@cherryactive.com, £10.00  

cherries  

at 2022-03-22 15:00:32.412267  

Verified: True  

----  

Transaction ID = 16  

Group: 11  

mreymacia@gmail.com -> kezza@cherryactive.com, £5.00  

cherries  

at 2022-03-22 15:01:56.175138  

Verified: True  

----  

(venv) tcassar@ubuntu:~/projects/settle$ settle simplify 11  

Group Password:  

(venv) tcassar@ubuntu:~/projects/settle$ settle verify -g 11
----  

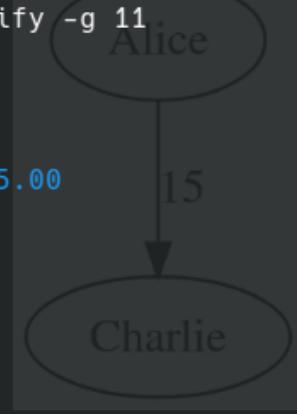
Transaction ID = 20  

Group: 11  

cassar.thomas.e@gmail.com -> kezza@cherryactive.com, £15.00  

at 2022-03-22 15:33:56.712326  

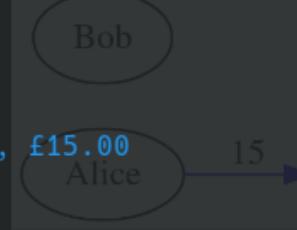
Verified: False
```



However, while hand correct algorithm.



Running Edmonds-Karp algorithm gives the max-flow of 11, weight of 15, and remaining edges.



The transaction that is generated is marked as unverified. This is because the sever does not have access to the private keys of the users. If the server held user private keys, it would not be a convincing security solution to say the least!

This example fulfils requirements **D1.6**, **D2.1**, **D2.4**, and **D2.5**

A note on UI: the unverified flashes. This is hard to put in a screenshot.

Hence, I can show that every requirement in section A and B has been met.

Evidence of Meeting Requirements - Section C

To show that my technical solution does in fact communicate over a REST API, I will show a simple get request made by the client and the corresponding logs produced by the server.

```

settle : settle-server
(venv) tcassar@ubuntu:~/projects/settle/src/server$ settle-server start -h 0.0.0.0
* Serving Flask app 'endpoint' (lazy loading)
* Environment: production
  (the environment variable 'FLASK_ENV' can be set to 'development' to change this)
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
WARNING:werkzeug: * Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
INFO:werkzeug: * Running on http://192.168.109.142:5000/ (Press CTRL+C to quit)
INFO:werkzeug:127.0.0.1 - - [22/Mar/2022 18:11:22] "GET /user/cassar.thomas.e@gmail.com HTTP/1.1" 200 -
[    ] with a capacity, and edges should have a notion of flow and unused capacity
settle : bash
(venv) tcassar@ubuntu:~/projects/settle$ settle whois cassar.thomas.e@gmail.com
[    ] A way to get the bottleneck value of a path, given a path of nodes
Found user with email cassar.thomas.e@gmail.com:
[    ] 3) A reliable recursive BFS that works on flow graphs
Name: Tom Cassar
Email: cassar.thomas.e@gmail.com
Modulus: 0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f46
65bda3ca30d767baedb969d1ee532ad92c8d1388ec09d5553db32605785db7e3fc9aaeeb1e4235d8d5038bf0467615a7e84442ff74e
c0d952598174dfadab34908e99b2bc8746918752dfa08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beaefcf0779dcde5569da8a776
376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb2754049facbd1efbaa6fc3b0a6e74fc70fe984a85ac7e548c833da1fa40
cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f4c530b17a5046eeb21652cf55c605,
Public Exponent: 0x10001
[    ] 5) Simplifying an entire graph using Edmonds Karp, using the method laid out in Settling a
\(venv\) tcassar@ubuntu:~/projects/settle\$ 
```

Here, it is possible to see that I have configured the API to run on the

<http://192.168.109.142:5000>.

The `settle whois <email>` command returns a user's name and public/private keys given an email. This shows that the server is accessible to the client via a REST api. There is a visible `GET` request, followed by an endpoint of the API. This satisfies **C1**

To show the program fulfilling **C2**, (handling exception codes gracefully), here is what happens when you attempt to `whois` an email that does not exist.

```

settle : settle-server
* Serving Flask app 'endpoint' (lazy loading)
* Environment: production
  (the environment variable 'FLASK_ENV' can be set to 'development' to change this)
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
WARNING:werkzeug: * Running on all addresses.
  WARNING: This is a development server. Do not use it in a production deployment.
INFO:werkzeug: * Running on http://192.168.109.142:5000/ (Press CTRL+C to quit)
INFO:werkzeug:127.0.0.1 - - [22/Mar/2022 18:11:22] "GET /user/cassar.thomas.e@gmail.com HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [22/Mar/2022 18:16:07] "GET /user/invalid@example.com HTTP/1.1" 404 -
[    ] with a capacity, and edges should have a notion of flow and unused capacity
settle : bash
(venv) tcassar@ubuntu:~/projects/settle$ settle whois cassar.thomas.e@gmail.com
[    ] A way to get the bottleneck value of a path, given a path of nodes
Found user with email cassar.thomas.e@gmail.com:
[    ] 3) A reliable recursive BFS that works on flow graphs
Name: Tom Cassar
Email: cassar.thomas.e@gmail.com
Modulus: 0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f46
65bda3ca30d767baedb969d1ee532ad92c8d1388ec09d5553db32605785db7e3fc9aaeeb1e4235d8d5038bf0467615a7e84442ff74e
c0d952598174dfadab34908e99b2bc8746918752dfa08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beaefcf0779dcde5569da8a776
376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb2754049facbd1efbaa6fc3b0a6e74fc70fe984a85ac7e548c833da1fa40
cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f4c530b17a5046eeb21652cf55c605,
Public Exponent: 0x10001
[    ] 5) Simplifying an entire graph using Edmonds Karp, using the method laid out in Settling a
\(venv\) tcassar@ubuntu:~/projects/settle\$ settle whois invalid@example.com
ERROR: Could not find requested resource on server, "User data invalid"
\(venv\) tcassar@ubuntu:~/projects/settle\$ 
```

The server has responded with a 404 error code to the client's GET request. On the client side, this has been detected and raised. It has then been handled gracefully before showing the client the type of error (a resource not found), and what that might be (invalid user info).

This type of behaviour is implemented for all errors that are expected to be raised during the programs normal operation. More severe errors, such as the warning when the user attempts to double sign a transaction, are coloured in red.

```
(venv) tcassar@ubuntu:~/projects/settle$ settle sign 12 ./src/crypto/sample_keys/t_private-key.pem
Email: cassar.thomas.e@gmail.com
Password:
      Key validated against server
Authorisation Error; aborting...
Failed to sign transaction: 12
Cannot overwrite signature of src
(venv) tcassar@ubuntu:~/projects/settle$
```

Above, it was shown that the client connects to the API using HTTP. This partially fulfills requirement **C3**. To fulfill it, it must be shown that JSON is used to communicate between client and server. Thus, I have briefly modified the `whois` command to dump the raw server response to STDOUT so that I can prove that this objective has been met (it was reverted after the test)

```
(venv) tcassar@ubuntu:~/projects/settle$ settle whois cassar.thomas.e@gmail.com
{'name': 'Tom Cassar', 'id': 20, 'modulus': '0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a759603f71e26f55bb64ca279500f4665bda3ca30d767baedb969d1ee532ad92c8d1388ec09d5553db32605785db7e3fc9aaeeb1e4235d8d5038bf0467615a7e84442ff74ec0d952598174dfadab34908e99b2bc8746918752cfa08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beafcfc0779dcde5569da8a776376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb2754049facbd1efbaa6fc3b0a6e74fc70fe984a85ac7e548c833da1fa40cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f4c530b17a5046eeb21652cf55c605', 'password': "b'\\xfb\\\\x00\\\\x1d\\\\xfc\\\\xff\\\\xd1\\\\xc8\\\\x99\\\\xf3')xq@bB\\\\xf0\\\\x97\\\\xa\\xe\\\\xcf\\\\xaSB\\\\xcc\\\\xf3\\\\xeb\\\\xcd\\\\x11aF\\\\x18\\\\x8eK'", 'email': 'cassar.thomas.e@gmail.com', 'pub_exp': '0x10001'}
00 and 500 codes gracefully
Found user with email cassar.thomas.e@gmail.com:
Client and Server should communicate over HTTP, using JSON as an information
interchange format
Name: Tom Cassar
Email: cassar.thomas.e@gmail.com
Modulus: 0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a759603f71e26f55bb64ca279500f4665bda3ca30d767baedb969d1ee532ad92c8d1388ec09d5553db32605785db7e3fc9aaeeb1e4235d8d5038bf0467615a7e84442ff74ec0d952598174dfadab34908e99b2bc8746918752cfa08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beafcfc0779dcde5569da8a776376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb2754049facbd1efbaa6fc3b0a6e74fc70fe984a85ac7e548c833da1fa40cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f4c530b17a5046eeb21652cf55c605,
Public Exponent: 0x10001
A transaction is tampered with in the database, it should be classed as unverified
(venv) tcassar@ubuntu:~/projects/settle$
```

This clearly shows that JSON is the information interchange format being used.

The screenshots of my CLI I have provided, I believe, effectively demonstrate a clear, easy to use command line interface. Thus, **C4** is also satisfied. Many more examples of output will be shown in the next section.

I can therefore say I have completed all of my requirements in Section C.

Evidence of Meeting Requirements - Section D

Ensuring the validity of transactions

Upon inspecting transaction 13

```
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -t 13
-----
Transaction ID = 13
Group: 11
cassar.thomas.e@gmail.com -> mreymacia@gmail.com, £12.87
chickens to simplify
at 2022-03-22 12:41:46.332096
Verified: True

(venv) tcassar@ubuntu:~/projects/settle$
```

Changing the amount owed in the database

id	pair_id	group_id	amount	src_key
2	11	3	1299	
3	23	3	1299	
4	24	3	1000	
5	24	3	1000	
6	24	3	1000	
7	24	3	1500	
8	24	3	1500	
9	32	3	500	
10	24	3	1299	
11	34	3	1387	
12	34	3	1387	
13	34	11	1490	

(1287 → 1490)

Querying again (here, the output of the old query is included first)

```
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -t 13
-----
Transaction ID = 13
Group: 11
cassar.thomas.e@gmail.com -> mreymacia@gmail.com, £12.87
chickens to simplify
at 2022-03-22 12:41:46.332096
Verified: True

(venv) tcassar@ubuntu:~/projects/settle$ settle verify -t 13
-----
Transaction ID = 13
Group: 11
cassar.thomas.e@gmail.com -> mreymacia@gmail.com, £14.90
chickens to simplify
at 2022-03-22 12:41:46.332096
Verified: False

(venv) tcassar@ubuntu:~/projects/settle$
```

Hence, **D1.1** is shown to be fulfilled.

Note that this transaction is signed. If I re-tamper with it to return the amount to 1287, it becomes valid again. I will show an attempt to re-sign it.

```
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -t 13
----  

Transaction ID = 13  

Group: 11  

cassar.thomas.e@gmail.com -> mreymacia@gmail.com, £12.87  

chickens to simplify  

at 2022-03-22 12:41:46.332096  

Verified: True  

----  

(venv) tcassar@ubuntu:~/projects/settle$ settle sign 13
Usage: settle sign [OPTIONS] TRANSACTION_ID KEY_PATH
Try 'settle sign --help' for help.

Error: Missing argument 'KEY_PATH'.
(venv) tcassar@ubuntu:~/projects/settle$ settle sign 13 ./src/crypto/sample_keys/t_private-key.pem
Email: cassar.thomas.e@gmail.com
Password: pre-settled
      Key validated against server
Authorisation Error; aborting...
Failed to sign transaction: 13
Cannot overwrite signature of src
(venv) tcassar@ubuntu:~/projects/settle$
```

Integrated requirements for how the end system handles settled transactions

- 1) Ensuring the validity of transactions
 - 1) If a transaction is tampered with in the database, it should be rejected
 - 2) A user should not be able to sign an already signed transaction
 - 3) A user should not be able to sign a transaction where they are not a member
 - 4) A user should not be able to sign a transaction with a key that does not belong to their account
 - 5) A user should not be able to sign a transaction without entering the password correctly
 - 6) Every transaction will have a unique identifier that has been verified by the server using the RSA sig/verif scheme from step 1
 - 7) A user should not be able to link a transaction to a group which is not part of the group
- 2) Ensuring that the debt simplification feature works
 - 1) All transactions in the group being settled should be verified

To demonstrate how the CLI handles invalid arguments, I attempted to sign an invalid transaction without a key. The CLI prompted me accordingly, and offered me the choice of using a `--help` flag.

D1.2 is thus fulfilled.

Below, I try to sign a transaction between `cassar.thomas.e@gmail.com` and `kezza@cherryactive.com` as a user on the account held by `mreymacia@gmail.com`. This is not allowed, fulfilling D1.3

```
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -t 5
----  

Transactions in the group being settled should be verified before the database
Transaction ID = 5  

Group: 3  

Sign Flow Diagram  

kezza@cherryactive.com -> cassar.thomas.e@gmail.com, £10.00  

test  

at 2022-03-21 10:44:16.839319  

Verified: False  

Simplify Flow Diagram  

Email provided doesn't match any of the users in the transaction
(venv) tcassar@ubuntu:~/projects/settle$
```

Integrated requirements for how the end system handles settled transactions

- 2) It should not be possible to simplify a group if there are unverified transactions in the group
- 3) The transaction structure of the group does not change, unless the group is simplified
- 4) The simplification should accurately simplify a system of debts so that each party is owed / owes a different amount of money after simplification
- 5) The simplifying process should result in unverified transactions

Integrated requirements for how the end system handles settled transactions

- 1) Users should be able to register for an account, providing a name and email address

Next, I try to sign the transaction as `cassar.thomas.e@gmail.com`, but with a different key to the one I registered my account with, fulfilling D1.4

```
(venv) tcassar@ubuntu:~/projects/settle$ settle sign 5 ./src/crypto/sample_keys/m_private-key.pem
Email: cassar.thomas.e@gmail.com
Password:  

Authorisation Error; aborting...
Private key provided does not match the listing in the db these transactions should be created as unsigned
(venv) tcassar@ubuntu:~/projects/settle$
```

Integrated requirements for how the end system handles settled transactions

- 1) Users should be able to register for an account, providing a name and email address
- 2) PEM formatted private key
- 3) Users should be able to create transactions where they are the author
- 4) Users should be able to create transactions where they are the receiver
- 5) These transactions should be created as unsigned
- 6) Users should be able to create a group with a name and password

I attempt to sign the transaction with my private key, but I mistype my password D1.5

```
(venv) tcassar@ubuntu:~/projects/settle$ settle sign 5 ./src/crypto/sample_keys/t_private-key.pem
Email: cassar.thomas.e@gmail.com
Password:  

Authorisation Error; aborting...
Password Incorrect
(venv) tcassar@ubuntu:~/projects/settle$
```

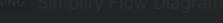
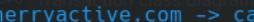
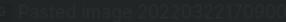
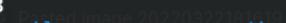
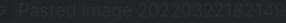
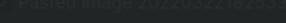
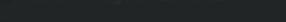
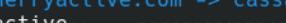
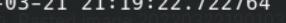
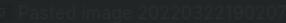
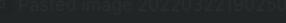
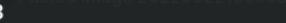
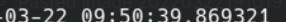
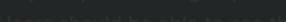
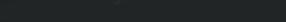
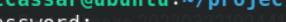
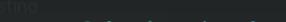
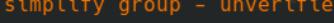
Integrated requirements for how the end system handles settled transactions

- 3) Users should be able to create a group with a name and password
- 4) Users should be able to join a group by group ID
- 5) Users should be able to mark a transaction as settled; transactions marked as settled when both parties involved mark the transaction as settled
- 6) Users should be able to see which groups they are a member of

D1.6 was shown previously.

D2.1, D2.4 and **D2.5** were shown previously.

To show that I have fulfilled **D2.2**, I will attempt to settle a group with unverified transactions

```
----  
Transaction ID = 7  
Group: 3  5) A user should not be able to sign a transaction without entering their account  
kezza@cherryactive.com -> cassar.thomas.e@gmail.com, £15.00  
test transaction IDs  
at 2022-03-21 10:49:25.471198  
  
  
----  
Transaction ID = 8  
Group: 3  6) Every time a transaction is pulled from the database and sent to be verified by the server using the RSA sig/verif scheme from sect  
kezza@cherryactive.com -> cassar.thomas.e@gmail.com, £15.00  
3   
at 2022-03-21 10:50:02.667656  
Evaluation  
  
----  
Transaction ID = 9 20220322181211  
Group: 3  2) Ensuring that the debt simplification feature works  
adhis@gmail.com -> cassar.thomas.e@gmail.com, £5.00  
scarn   
at 2022-03-21 13:09:25.570672  
  
  
----  
  
----  
Transaction ID = 10 20220322184913  
Group: 3  1) Users should be able to register for an account, providing name and PEM formatted private key  
kezza@cherryactive.com -> cassar.thomas.e@gmail.com, £12.99  
cherry active   
at 2022-03-21 21:19:22.722764  
  
  
----  
  
Transaction ID = 11 20220322190508  
Group: 3  2) Users should be able to create transactions where they are the debtor; these transactions should be created as unsigned  
cassar.thomas.e@gmail.com -> mreymacia@gmail.com, £13.87  3) Users should be able to create a group with a name and password  
chickens   
at 2022-03-22 09:50:39.869321  
  
(venv)  4) Users should be able to join a group by group ID  
tcassar@ubuntu:~/projects/settle$ settle simplify 3  
Group Password:   
Problem settling group...   
This action was not allowed by the server, "Couldn't simplify group - unverified transactions in group"   
(venv) tcassar@ubuntu:~/projects/settle$ 5) User information
```

As aforementioned, the **Verified: False** messages blink. This is why they are not shown in this screenshot (evidence of blinking in video provided).

To show that I satisfy **D2.3, D3.2, D3.3, D3.4**, and **D3.10** I will make a new group and add two users to it. I will then add a single transaction and attempt to settle. One transaction cannot be simplified, thus we should be warned that simplification cannot happen

```
(venv) tcassar@ubuntu:~/projects/settle$ settle new-group
Name: test d1.7
Password:
Repeat for confirmation:
"Created group ID=12 named test d1.7"

You can join this group with `settle join`  

(venv) tcassar@ubuntu:~/projects/settle$ settle join 12
Email: cassar.thomas.e@gmail.com
Your password:
Group Password:
Successfully joined group 12
(venv) tcassar@ubuntu:~/projects/settle$ settle join 12
Email: kezza@cherryactive.com
Your password:
Group Password:
Authorisation Error; aborting...
Password Incorrect
(venv) tcassar@ubuntu:~/projects/settle$ settle join 12
Email: kezza@cherryactive.com
Your password:
Group Password:
Successfully joined group 12
```

D3.3 and **D3.4** have been fulfilled

Making a transaction (thus fulfilling **D3.2** and **D3.10**)

```
(venv) tcassar@ubuntu:~/projects/settle$ settle new-transaction
Email of payee: kezza@cherryactive.com
Amount (in GBP): 28
Reference: test one transaction in a group
Group: 12
Your email: cassar.thomas.e@gmail.com
Password:
Transaction generated with ID=24
Sign with `settle sign 24`
(venv) tcassar@ubuntu:~/projects/settle$ █
```

It is initially unsigned - I then sign it with both users

```
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -t 24
-----
Transaction ID = 24
Group: 12
cassar.thomas.e@gmail.com -> kezza@cherryactive.com, £28.00
test one transaction in a group
at 2022-03-22 19:29:48.330112
Verified: False

(venv) tcassar@ubuntu:~/projects/settle$ settle sign 24 ./src/crypto/sample_keys/d_private-key.pem
Email: kezza@cherryactive.com
Password:
    Key validated against server
    successfully signed transaction
    Successfully appended signature in database!
(venv) tcassar@ubuntu:~/projects/settle$ settle sign 24 ./src/crypto/sample_keys/t_private-key.pem
Email: cassar.thomas.e@gmail.com
Password:
    Key validated against server
    successfully signed transaction
    Successfully appended signature in database!
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -t 24
-----
Transaction ID = 24
Group: 12
cassar.thomas.e@gmail.com -> kezza@cherryactive.com, £28.00
test one transaction in a group
at 2022-03-22 19:29:48.330112
Verified: True
```

It is now verified. Next, I try to settle group 12. Since it only one transaction, I should be alerted that no changes were made.

```
(venv) tcassar@ubuntu:~/projects/settle$ settle simplify 12
Group Password:
No changes were made
"No changes made to debt structure - heuristic did not find anywhere to simplify"
(venv) tcassar@ubuntu:~/projects/settle$
```

Indeed, I am.

To show **D3.1**, I will register an account, first providing it with a public key. It should reject this. Then, I will then register the account with the correct key. I will confirm the creation of the account with the `settle whois` command, thus also fulfilling **D3.9**

```
(venv) tcassar@ubuntu:~/projects/settle$ settle register
Full Name: example person
Email: example@gmail.com
Password:
Repeat for confirmation:
Path to RSA key: /home/tcassar/projects/settle/src/crypto/sample_keys/d_public-key.pem
unable to load Private Key
139870883080000:error:0909006C:PEM routines:get_name:no start line:../crypto/pem/pem_lib.c:745:Expecting: ANY PRIVATE KEY
Failed to create account - issue with given RSA key;
File not in PEM private key format
(venv) tcassar@ubuntu:~/projects/settle$
```

Successful creation

```
(venv) tcassar@ubuntu:~/projects/settle$ settle register
Full Name: example person
Email: example@gmail.com
Password: 
Repeat for confirmation:
Path to RSA key: /home/tcassar/projects/settle/src/crypto/sample_keys/t_private-key.pem
Account created successfully

Name: example person
Email: example@gmail.com
Modulus: 0xc26c13c82f5df38ebef77256144a471dfb1f62ff78faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30d767baed
b969d1ee532ad92c8d1388ec09d553db32605785db7e3fc9aeeb1e4235d8d5038bf0467615a7e84442ff74ec0d952598174dfadab34908e99b2bc874691
8752cfa08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beafc0f779dcde5569da8a776376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb275404
9facbd1efbaa6fc3b0a6e74fc70fe984a85ac7e548c833da1fa40cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f
4c530b17a5046eeb21652cf55c605,
Public Exponent: 0x10001

(venv) tcassar@ubuntu:~/projects/settle$ settle whois example@gmail.com
Found user with email example@gmail.com:
Name: example person
Email: example@gmail.com
Modulus: 0xc26c13c82f5df38ebef77256144a471dfb1f62ff78faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30d767baed
b969d1ee532ad92c8d1388ec09d553db32605785db7e3fc9aeeb1e4235d8d5038bf0467615a7e84442ff74ec0d952598174dfadab34908e99b2bc874691
8752cfa08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beafc0f779dcde5569da8a776376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb275404
9facbd1efbaa6fc3b0a6e74fc70fe984a85ac7e548c833da1fa40cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f
4c530b17a5046eeb21652cf55c605,
Public Exponent: simple 0x10001

(venv) tcassar@ubuntu:~/projects/settle$
```

To show fulfillment of **D3.5**, **D3.7**, and **D3.8**. I will find an open transaction and mark it as settled. It should no longer appear in a set of group debts after both parties mark it as settled, but should appear if only one party has marked it as settled.

```
(venv) tcassar@ubuntu:~/projects/settle$ settle show -t
Email: keith@npl.com

You owe cassar.thomas.e@gmail.com £12.99

Reference: scran2
Agreed upon at 2022-03-19T17:05:18.172694ID: 3
Unverified

cassar.thomas.e@gmail.com owes you £12.99

Reference: scran
Agreed upon at 2022-03-19T16:53:37.720130ID: 2
Unverified
-----
You owe and are owed nothing; all debts settled
Your unverified totals => all debts settled
(venv) tcassar@ubuntu:~/projects/settle$
```

I will now sign transaction 2 by both parties - this has already been documented and thus won't be shown again.

Showing group three (and fulfilling **D3.8**), we see two transactions. After one party marks it as settled, it is still classed as an open transaction. Once the second party has ticked it off it is counted as settled. Hence, it is no longer shown with the rest of the open group transactions.

```
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -g 3
-----
Transaction ID = 2
Group: 3
cassar.thomas.e@gmail.com -> keith@npl.com, £12.99
scran
at 2022-03-19T16:53:37.720130
Verified: True
-----
Transaction ID = 3
Group: 3
keith@npl.com -> cassar.thomas.e@gmail.com, £12.99
scran2
at 2022-03-19T17:05:18.172694
Verified: False

(venv) tcassar@ubuntu:~/projects/settle$ settle tick 2
Email: cassar.thomas.e@gmail.com
Password:
Transaction 2 marked as settled!
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -g 3
-----
Transaction ID = 2
Group: 3
cassar.thomas.e@gmail.com -> keith@npl.com, £12.99
scran
at 2022-03-19T16:53:37.720130
Verified: True
-----
Services
-----
Transaction ID = 3
Group: 3
keith@npl.com -> cassar.thomas.e@gmail.com, £12.99
scran2
at 2022-03-19T17:05:18.172694
Verified: False

(venv) tcassar@ubuntu:~/projects/settle$ settle tick 2
Email: keith@npl.com
Password:
Transaction 2 marked as settled!
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -g 3
-----
Transaction ID = 3
Group: 3
keith@npl.com -> cassar.thomas.e@gmail.com, £12.99
scran2
at 2022-03-19T17:05:18.172694
Verified: False

(venv) tcassar@ubuntu:~/projects/settle$
```

16 rows retrieved, starting from 1 in 50 ms (execution: 5 ms, fetching: 45 ms)

This shows transactions 2 and 3 listed as unmarked until both parties have ticked transaction 2. In the final call to `verify`, only transaction 3 is shown. Hence, **D3.5** and **D3.7** have been met.

Evidence for **D3.6**:

```
(venv) tcassar@ubuntu:~/projects/settle$ settle show -g
Email: cassar.thomas.e@gmail.com          4      1000
Groups that you are a member of:
Group:                                     4      1500
                                         d_private-key.pem    7      1500
                                         Name: test group, ID = 3    4      500
Group:                                     4      1299
                                         m_private-key.pem    9      1299
                                         Name: simplify test, ID = 11    4      1387
Group:                                     4      1387
                                         m_public-key.pem   10      1387
                                         Name: test, ID = 8    11      1387
Group:                                     4      1287
                                         n_public-key.pem   12      1287
                                         Name: test d1.7, ID = 12    11      500
                                         t_init_py        14      1000
                                         500

(venv) tcassar@ubuntu:~/projects/settle$
```

Evidence of Meeting Requirements - Section E

To show that I have implemented the database structure that I laid out in my requirements, I will screenshot the database tables, and briefly comment on which requirements each table fulfils.

Not only does this show the structure of the data being stored, it also proves that I am storing data as per my requirements.

Transactions

This table fulfills **E2.1 → E2.8** in conjunction with the `pairs` table and the `keys` table. It also satisfies **E3.4**

src_settled	dest_settled
a6ccb413d1898c8f678c7eaa7673be5a...	1
d6fd278dcf395fa061a5892cc109302...	0
52d93117218a9bb622ec5963f2e415a3...	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
b5e5d63e99a53f9bc3598ff50573d6c0...	0
f3dcfc86e43bbfcfa9d8e54fb33fa539...	1
31b2168e1c659a5832c5f88f5667492...	0
e7c57010eea8f4aa8162f35271cb57a...	1
b9c636f2adbff4aab3a96a636619d724...	1
d3cffce6ccbf3591b8f98a1cc94ba4c01...	1
ab51f621d3207ab373f6d59bbe4e54f...	0

amount	src_key	dest_key	reference	time_of_creation	src_sig	dest_sig
1299	4	5	scran	2022-03-19T16:53:37.720130	0x6619229faa043a839cc3595bf1284da28b58aa95b4e...	0xae6de6b18256
1299	5	4	scran2	2022-03-19T17:05:18.172694	0x5abbc6445a38	0x9e0d44e0e8fa
1000	9	4	security transaction	2022-03-21 10:05:48.828933		
1000	9	4	test	2022-03-21 10:44:16.839319		
1000	9	4	kez money	2022-03-21 10:47:25.463493		
1500	9	4	test transaction IDs	2022-03-21 10:49:25.471198		
1500	9	4	3	2022-03-21 10:50:02.667656		
500	10	4	scarn	2022-03-21 13:09:25.570672		
1299	9	4	cherry active	2022-03-21 21:19:22.722764		
1387	4	8	chickens	2022-03-22 09:50:39.869321		
1387	4	8	chickens	2022-03-22 09:55:42.8000220	0x30a31c8375275d778542c9332f56e0232b83fc22750...	0xb1da9bcfe0000
1287	4	8	chickens to simplify	2022-03-22 12:41:46.332096	0x8390075842746e0cb35b010118478dffccfc670311c6...	0x5f7fbabdf263
500	4	8	simplify test	2022-03-22 14:57:10.711745	0x249baaf2517307987f154619a99f136109357b285d6...	0x75d4f3e11805
1000	4	9	cherries	2022-03-22 15:00:32.412267	0x4bf53cf1ab337b59dbf69e82905841f891a4c5dad4...	0x7f68499aed51
500	8	9	cherries	2022-03-22 15:01:56.175138	0x1fda9bf10034dad2689f4285b9af7de75adfa69616b...	0x77decc27c32f
2800	4	9	test one transaction	2022-03-22 19:29:48.330112	0xaaa65d3ddd7dbd1baa1da6d74d6c87b778a59fe2354...	0x71c2db2484b1

	pair_id	group_id
id		
1	2	11
2	3	23
3	4	24
4	5	24
5	6	24
6	7	24
7	8	24
8	9	32
9	10	24
10	11	34
11	12	34
12	13	34
13	14	34
14	15	38
15	16	39
16	24	38
17		

Pairs

	id	src_id	dest_id
1		11	20
2		23	21
3		24	25
4		32	20
5		34	20
6		38	20
7		39	24
			25

Keys

	id	n	e
1	3	0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30...	0x10001
2	4	0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30...	0x10001
3	5	0xb2c18e327280ff4abe3ed337b022518563a336871851178f9ee13cd8085c875c080184d836ca0a3f2d1fb771c98931cdbe2...	0x10001
4	6	0xb2c18e327280ff4abe3ed337b022518563a336871851178f9ee13cd8085c875c080184d836ca0a3f2d1fb771c98931cdbe2...	0x10001
5	7	0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30...	0x10001
6	8	0xb2c18e327280ff4abe3ed337b022518563a336871851178f9ee13cd8085c875c080184d836ca0a3f2d1fb771c98931cdbe2...	0x10001
7	9	0xab3cec18ae8df68c502ed71ce3376cca00c5e82aeebf9aa504794ed2b4b340093e381cbe2dc84c274da640a2d2c379a8c47...	0x10001
8	10	0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30...	0x10001
9	11	0xc26c13c82f5df38ebef77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30...	0x10001

Users

The users table, along with the keys table above, fulfil requirements **E1.1 → E1.5**

	key_id
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11

name	email	password
admin	admin@admin.com	b'\x{fb}\x{00}\x{1d}\x{ff}\x{fc}\x{ff}\x{d1}\x{c8}\x{99}\x{f3})x{q}@b{B}\x{f0}\x{97}\x{ae}\x{cd}\x{cc}\x{f3}\x{eb}\x{cd}\x{1a}\x{f}\x{18}\x{8e}\x{eK}'
Cassar	cassar.thomas.e@gmail.com	b'\x{fb}\x{00}\x{1d}\x{ff}\x{fc}\x{ff}\x{d1}\x{c8}\x{99}\x{f3})x{q}@b{B}\x{f0}\x{97}\x{ae}\x{cf}\x{1a}\x{f}\x{18}\x{8e}\x{eK}'
Dunes Tahir	Keith@npl.com	b'\x{90}\x{9d}\x{a6}\x{6u}\x{c6}<\x{04}\x{92}\x{14}\x{7}\x{86}\x{bd}\x{ec}\x{b2}\x{01}\x{H}\x{act}\x{TB}\x{de}\x{d}=M\x{ab}\x{d1}\x{f2}\x{f8}'
oo	foo@bar.com	b'\v\x{d3}\x{bc}\x{A}\x{c9}\x{f5}\x{88}\x{f7}\x{fc}\x{d6}\x{d5}\x{bf}\x{f6}\x{18}\x{f8}\x{f8}\x{8K}\x{1c}\x{ca}\x{b2}\x{68}\x{82}\x{p1}\x{00}\x{b9}\x{eb}\x{94}\x{13}\x{80}\x{e}'
oo Bar	foobar@example.com	b'\t#\x{H}\x{07}\x{e4}\x{af}\x{85}\x{f1}\x{f}\x{b4}\x{8e}\x{e3}\x{bc}\x{a8}\x{9d}\x{ff}\x{d1}\x{f1}\x{6Y}\x{f9}\x{f9@\x{a2}\x{b1}{\x{0b}\x{a2}\x{b1}\x{0b}\x{8c}\x{ck}\x{c5}}'
aia Rey Macia	mreymacia@gmail.com	b'\x{8}\x{ad}\x{08}\x{8b}\x{5S}\x{81}>\x{8e}\x{0p}/\x{f6}\x{16}\x{0}(\x{97}\x{cd}\x{d1}\x{f1}\x{6o}\x{89}\x{n}\x{0}\x{87}\x{^}\x{l}'
ez Carey	Kezza@cherryactive.com	b'\x{ff}\x{8e}\x{e7}\x{b3}\x{1f}\x{12}\x{1e}\x{eb}\x{be}\x{46}\x{wZ}\x{81};\x{f4}`\x{bf}\x{d1}\x{d8}\x{d0}\x{e4}\x{9d}\x{H}\x{k}\n\x{14L}\x{cf}\x{~}'
hish	adhish@gmail.com	b'\x{f5}\x{1a}\x{fe}\x{17}\x{c9}\x{e1}\x{85}\x{c9}\x{ec}\x{b9}\x{9ao}\x{v}\x{d1}/\x{19K}\x{ec}\x{e2m}\x{19c}\x{b9}\x{tr}\x{86}\x{8e}\x{J}\x{9c}\x{f4}\x{9bp}'
sample person	example@gmail.com	b'\x{98=i/d}\x{81}\x{fe}\x{bemo}\x{a6}\x{07}\x{c}\n\x{e6}\x{86I}\x{ff7}\x{e6}\x{9F}\x{80}\x{t}\x{06}\x{e4}\x{fa}\x{db}'

	19	ad
	20	Tc
	21	Yc
	22	fc
	23	Fc
	24	Ma
	25	Kc
	26	ac
	27	e>
1	2	3
4	5	6
7	8	9

Groups

The final outstanding requirements in Section E are met by the `groups` table and `groups_link` table, meeting **E3.1 → E3.3**

The screenshot shows two database tables:

- groups** table:

ID	Name	Password
1	3 test group	b"6\xf0(X\x0b\xb0,\xc8'*\x9a\x02\x0fB\x00\xe3F\xe2v\xaeFNE\xee\x80tUt\xe2\xf5\xab\x80"
2	6 test2	b'1\xf0\x18\x95\xdf\x02;0\xcbH\xf6\xeb\xfe\xc3~\x9d!Va\xa3.\xa1\x0f\xfa\x17k \xf2~tn8'
3	ryans mandem	b'g\xad\x03\x8d\xf7\xd7^U\xd8n\x15z\xdb\x14\xed\xdc\xd9\xa6\\$.\xee\xaz\xb3\x85\xab\xb5V\xbd\xb7\xdcc1'
4	8 test	b"6\xf0(X\x0b\xb0,\xc8'*\x9a\x02\x0fB\x00\xe3F\xe2v\xaeFNE\xee\x80tUt\xe2\xf5\xab\x80"
5	9 test	b"6\xf0(X\x0b\xb0,\xc8'*\x9a\x02\x0fB\x00\xe3F\xe2v\xaeFNE\xee\x80tUt\xe2\xf5\xab\x80"
6	10 Foo's Test Group	b'1#\x07\xe4\xaf\x85\xf1\xb4\x8e\xe3\xbc\xab\xd\xff\xd1\xf1#6Y\xf9\xf90\xab\xb1{\x0b\x8c\xc5'
7	11 simplify test	b"6\xf0(X\x0b\xb0,\xc8'*\x9a\x02\x0fB\x00\xe3F\xe2v\xaeFNE\xee\x80tUt\xe2\xf5\xab\x80"
8	12 test d1.7	b"6\xf0(X\x0b\xb0,\xc8'*\x9a\x02\x0fB\x00\xe3F\xe2v\xaeFNE\xee\x80tUt\xe2\xf5\xab\x80"

- group_link** table:

ID	group_id	usr_id
1	16	3
2	17	3
3	18	10
4	19	3
5	20	3
6	21	11
7	22	11
8	23	8
9	24	11
10	25	11
11	26	12
12	27	12

Hence, I have entirely fulfilled every requirement I outlined in Section E, and have thus completed my project entirely.

Evaluation

I will evaluate my completed project by gaining the opinion of the end user that I talked in the Analysis phase. I plan to give him a demonstration of the project, and let him use the project for a week. I will then collect and reflect on his feedback.

In the demo, I thought it to be important to directly address the main concerns that he originally highlighted. These were mainly centred around system administrators (me) altering the amount of money that people owe each other.

To put his mind at ease, I gave him a live demonstration of the tampering test I used to show that my system fulfilled requirement **D1.1**. He was extremely impressed with this and spent the next 5 minutes tampering with data in the database, and watching the previously verified transactions become unverified.

Once he had convinced himself that public key cryptography works, we moved on to addressing his next concern - debt simplification. Again, I showed him the 'simplify debt' example that I used in my analysis, and we spent the next 10 minutes building graphs and watching them simplify down.

During this, he became increasingly comfortable using the CLI. He told me he was a bit hesitant when he saw it - it looked like nothing he had ever used before. While playing with the security and debt simplification features of the app, it was quite interesting to see just how quickly he got used to it. He told me that a CLI was definitely a good choice for this, due to just how simple it is to get things done.

However, he was not completely without criticism.

He reported, fairly, that it was slightly annoying to have to keep entering your email and password, especially when he had entered this information just one command before.

He also said that he would like a way to see his closed transactions in a list view, not just accessing them by ID.

Finally, he said that he was confused as to why he had to generate an OpenSSL RSA Private key himself and then feed it into the `settle register` command. In his opinion, a key generate command would have been helpful.

I think that these are entirely valid concerns, and would be where I go next if I were to continue to improve the project.

I decided to look into how I would go about implementing the end user's suggestions.

The email / password remembering could be achieved relatively straightforwardly with the `click` library using the concept of `contexts`. This is most definitely something that is doable, and would do a lot to aid the overall user experience.

Similarly, it would be trivial to add use a binding to OpenSSL and have my client be able to generate an RSA private key through the CLI. This is just something I hadn't considered in my Analysis of the project, but would definitely add to the user experience.

Adding a view for closed transactions would be another command a modified SQL statement, and I would probably need to add query parsing to my API endpoints, so that I could keep my server processes almost identical (aside from writing the new statement and query parser). I would then be able to pull transactions either open or closed, fully with code that already exists.

However, I was on the whole extremely happy with his response. When I asked him on how good a fit my solution to his problem was, he told me that with his additions, it would be absolutely perfect.

When considering the extent to which I met my own high level requirements, I am really quite pleased. While everything was planned and meticulously designed, I am still surprised at just how robust my end product is.

Individual Requirement Reflection

An RSA implementation that will allow the signing, and verification, of transactions

On the whole I have built a robust, functional implementation of RSA for this project, which effectively fulfils the aim of preventing tampering with transactions (as well as more menial things such as password hashing). If I were to do this project again, however, I would not have written the cryptography side of things in Python.

Due to the way that Python stores numbers, it is impossible to rule out the possibility of side channel attacks. Similarly, I did not use a constant time algorithm for the modular exponentiation step of RSA. Thus, it would be possible for an attacker to use a timing attack to deduce the private exponent in a user's private key.

I think in the context that I had intended the project to be used in, this is not currently a massively pressing issue. Knowing about it does mean, however, that I would like to protect against it even if it means learning a new programming language.

A way to settle the debts of the group in as few as possible (heuristically speaking) monetary transfers

I am most pleased with the debt simplification - I think that that is a genuinely useful idea, and I am happy that it works so well. I did learn some interesting things about how the heuristic model behaved when I was experimenting with making graphs with my end user.

I discovered that the algorithm that I implemented to simplify groups of debt works best on densely connected graphs as more augmenting paths can be found. This makes me wonder if there is a way to change how I search for augmenting paths in the flow graph to try and end up with longer chains of debt.

The problem is that a path of $n + 1$ edges has a higher chance of a smaller bottleneck value than a path with n edges, and could end up leading to the time complexity of the algorithm being dependent on flow. This would potentially make the algorithm less efficient.

This is something that I would like to investigate more in the future.

Another thing that I would like to test is adding cycle detection to the graph. I feel as though I could improve the performance of my heuristic if I simplified cycles before running any flow graph algorithms. However, this may worsen the time complexity of the process as a whole. This, it is another thing to experiment with.

A server-side component of the application which can verify transactions, and store / retrieve them from a database

A client-side component of the application that will have a simple user interface (CLI)

```
(venv) tcassar@ubuntu:~/projects/settle$ settle verify -t 5
  transactions in the group being settled should be verified
  the database
-----
Transaction ID = 5
Group: 3
kezza@cherryactive.com -> cassar.thomas.e@gmail.com, £10.00
test
at 2022-03-21 10:44:16.839319
Verified: False
  Simplify Flow Diagram
(venv) tcassar@ubuntu:~/projects/settle$ settle sign 5 ./src/crypto/sample_keys/m_private-key.pem
Email: mreymacia@gmail.com
Password:
Email provided doesn't match any of the users in the transaction
(venv) tcassar@ubuntu:~/projects/settle$
```

A screenshot of the simple, easy to use CLI

The next two requirements can be addressed as one. I am very happy with the client-server model of the system, even though the client-server model was much more work than I imagined. However, it is quite impressive to see communication across a network, even if it is currently a localhost network.

Having a thin client is particularly easy and effective as it means that, if more people were to adopt this solution and I were to set up a full-time server, this could be run on absolutely everything.

As was discussed with the end user, there are certain improvements that I could make to the CLI. These are very much quality of life improvements, and the CLI that I provided was more than adequate at fulfilling all of my initial requirements

A database that should be able to store user and transaction information

Finally, I am happy with my database. However, as I became more comfortable with SQL during the project, having had absolutely no experience with it before, I realise that I have a redundant relationship in my database design. The transaction table references the keys table. Having learned about the join statement, I now see that this link is redundant and ought not to be there.

```

1 # coding=utf-8
2
3 """
4 Unit tests for key handling
5 """
6
7 import os
8 from unittest import TestCase
9
10 from src.crypto import keys
11
12
13 class TestRSAKeyLoading(TestCase):
14     """Tests for RSA Keys"""
15
16     def setUp(self) -> None:
17         """Initialise loaders fresh between tests"""
18         self.loader = keys.RSAKeyLoader()
19         os.chdir("/home/tcassar/projects/settle/src")
20         print(os.system("pwd"))
21         self.key_path = "./crypto/sample_keys/
d_private-key.pem"
22         self.pub_key_path = "./crypto/sample_keys/
d_public-key.pe"
23
24     def test_file_loading(self):
25         """Tests that file is being loaded correctly
assuming correct file"""
26
27         expected_start = "modulus"
28         # test assumes if it starts off fine it will
continue being fine
29         received_start = self.loader.load(self.
key_path)
30         self.assertEqual(expected_start,
received_start[:7])
31
32     def test_parsing(self):
33         """Checks that keys are parsed correctly and
that errors are raised when inputs are incorrect"""

```

```
34         loader = self.loader
35
36         loader.load(self.key_path)
37         loader.parse()
38
39         # compare parsed file to known values from
39         # test keys.
40         # lay out known values
41         k_e = 65537
42
43         k_d =
44             4231860502610347555352698504423047563944256813822996
45             3925734038989576569874864579669508371562955146126547
46             8779552919144303233011283933461726857674426838187389
47             6721663847672076407494464781754007098969569658821095
48             1082558372250397158894945714319204774976709066583095
49             9656389449195952398626578010004019881519663179628663
50             1049344644422180019003905454847914081363056944472084
51             7689992964879750244948034517871603890465032993904165
52             597478235001604038074703682680413387037575013624446
53             8012222270131756222483185448064466203435454227528122
54             7314387046825525570376824278684460211954070416261823
55             5709118874694072838664235842554512315661905
56
57         k_n =
58             2161679203114375274630941557945232020251089312607308
59             7652383723408105063600070147761500936454570470862786
60             9702069833686361660030089751732511247633740543213460
61             3035445702142516535603778163693003610640441829541372
62             6431002556836900067049867944499904312842155745102543
63             7279819137427553642625019821057158620227234339857381
64             3906769437847646651445567701079686709061678948557745
65             8637370869261774337704654931841775536224420241750390
66             3181827421441408785602795322818434551136750209831842
67             5250366913849245188288359620277590091100050269612943
68             3770297061845035322423193568722584809589006962060587
69             421954603201784497846158263582144177430642603
70
71         # build lists for testing
72         cases = [
73             "pub_exp",
74             "priv_exp",
```

```
50             "mod",
51             "prime1",
52             "prime2",
53             "exponent1",
54             "exponent2",
55             "coefficient",
56         ]
57
58     known = [k_e, k_d, k_n]
59
60     key = keys.RSAPrivateKey(loader)
61
62     received = [
63         key.e,
64         key.d,
65         key.n,
66         key.p,
67         key.q,
68         key.exp1,
69         key.exp2,
70         key.crt_coef,
71     ]
72
73     for test, known_val, rec_val in zip(cases,
74                                         known, received):
75         with self.subTest(test):
76             self.assertEqual(rec_val, known_val)
77             self.assertEqual(type(rec_val), type
78                             (known_val))
79
80     def test_file_not_found(self):
81         path = "./adsads"
82
83         with self.assertRaises(keys.RSAParserError):
84             loader = self.loader
85             loader.load(path)
86
87     def test_wrong_format(self):
88         """Checks that we can deal with files being
89         the wrong format"""
90         with self.assertRaises(keys.RSAParserError):
```

```
88             self.loader.load("./crypto/sample_keys/  
d_public-key.pem")  
89  
90     def test_unparsed_key(self):  
91         """Check accessing attributes"""  
92         loader = self.loader  
93         loader.load(self.key_path)  
94         key = keys.RSAPrivateKey(loader)  
95  
96         with self.assertRaises(keys.RSAKeyError):  
97             _ = key.asdf  
98  
99     def test_unloaded_key(self):  
100        """Checks that error is raised if parse()  
is called on a loader object which has not yet  
loaded a key"""  
101        ldr = self.loader  
102        with self.assertRaises(keys.RSAParserError  
):  
103            ldr.parse()  
104  
105    def test_public_key(self):  
106        """Checks that public keys will raise an  
exception if they are used for signing"""  
107        ldr = self.loader  
108        ldr.load(self.key_path)  
109        ldr.parse()  
110  
111        pub_key = keys.RSAPublicKey(ldr)  
112  
113        with self.subTest("allowed access"):  
114            self.assertEqual(pub_key.e, 65537)  
115  
116        with (self.subTest("deny access"), self.  
assertRaises(keys.RSAPublicKeyError)):  
117            _ = pub_key.p  
118
```

```

1 # coding=utf-8
2 import hashlib
3 import sys
4 from unittest import TestCase
5
6 from src.crypto import hashes
7
8
9 class TestHash(TestCase):
10     """
11         Test hash interfaces (functionality not aim of
12             testing as functionality comes from hashlib
13             1) Check initialisation as expected
14             2) Check update
15             3) Check digest
16             4) Check hexdigest == intdigest
17             """
18
19     def test_init(self) -> None:
20         """Checks that _hasher is initialised
21             correctly i.e. no strange start values"""
22         h = hashes.Hasher()
23         self.assertEqual(
24             h.digest().int_digest(),
25             int.from_bytes(hashlib.sha3_256(b"").digest(),
26                           byteorder=sys.byteorder),
27         ) # should be initialised empty
28
29     def test_hash(self) -> None:
30         """tests that hash object can validate hash
31             looking numbers"""
32         with self.subTest("valid"):
33             h_val = hashlib.sha3_256(b"1234").digest()
34             h = hashes.Hash(h_val) # create a
35             # definitely valid hash
36
37             with self.subTest("too short"):
38                 with self.assertRaises(hashes.HashError
39                 ):
40                     hashes.Hash(b"12")

```

```
35
36         with self.subTest("too long"):
37             with self.assertRaises(hashes.HashError):
38                 hashes.Hash(
39                     b"
40                         1157920892373161954235709850086879078532699846656405
41                         640394575840079131296399360"
42
43         with self.subTest("wrong type"):
44             with self.assertRaises(hashes.HashError):
45                 # noinspection PyTypeChecker
46                 hashes.Hash("dave")
47
48     def test_update_digest(self) -> None:
49         """Ensures that a hash with a given value
50         will digest the correct thing"""
51         h = hashes.Hasher(b"1234")
52         h_ = hashlib.sha3_256(b"1234")
53
54         with self.subTest("before update"):
55             self.assertEqual(h.digest().h, h_.digest())
56
57             update_msg = b"test hash update"
58             h.update(update_msg)
59             h_.update(update_msg)
60
61         with self.subTest("after update"):
62             self.assertEqual(h.digest().h, h_.digest())
63
64     def test_hasher_fails(self):
65         """Checks that hasher objects when not bytes
66         are passed into it"""
67         with self.assertRaises(hashes.HasherError):
68             # noinspection PyTypeChecker
69             hasher = hashes.Hasher("abd")
```

```

1 # coding=utf-8
2
3 """
4 Testing sign / verify through RSA working as
5 expected
6 """
7 import os
8 from unittest import TestCase
9
10 from src.crypto import keys
11 from src.crypto import rsa
12 from src.transactions.transaction import Signable
13
14 def setUpModule():
15     os.chdir("/home/tcassar/projects/settle/src")
16
17
18 class TestRSA(TestCase):
19     """Just tests RSA parts"""
20
21     def setUp(self) -> None:
22         # load keys
23         ldr = keys.RSAKeyLoader()
24         ldr.load("./crypto/sample_keys/d_private-key
25 .pem")
26         ldr.parse()
27
28         # build public and private
29         self.private = keys.RSAPrivateKey(ldr)
30         self.public = keys.RSAPublicKey(ldr)
31
32     def test_encryption(self):
33         """Checks to see if process is reversible,
34         and encrypted is different to how it started"""
35         message = " | maia"
36         m_bytes = bytes(message, encoding="utf8")
37         encrypted = rsa.RSA.encrypt(m_bytes, self.
38             public)
39         # TODO: actually fix byte overflow affair
40         decrypted = rsa.RSA.bytes_to_str(rsa.RSA.

```

```
37    naive_decrypt(encrypted, self.private))
38
39        with self.subTest("Catch Public Key"):
40            with self.assertRaises(rsa.
41                DecryptionError):
42                _ = rsa.RSA.naive_decrypt(encrypted
43                , self.public) # type: ignore
44
45        with self.subTest("encrypted"):
46            self.assertNotEqual(m_bytes, encrypted)
47
48        with self.subTest("Successful decryption"):
49            self.assertEqual(message, decrypted)
50
51
52    def test_RSA_sign(self):
53        """Checks for consistent creating /
54        verifying of a 'signature'"""
55
56        message = " | maia"
57        m_bytes = message.encode("utf8")
58        sig: bytes = rsa.RSA.sign(m_bytes, self.
59        private)
60        de_sign: bytes = rsa.RSA.inv_sig(sig, self.
61        public)
62
63        self.assertEqual(m_bytes, de_sign)
```

```
1 # coding=utf-8
2 from unittest import TestCase
3
4 from src.simplify.base_graph import Digraph
5 from src.simplify.flow_graph import FlowGraph
6 from src.simplify.graph_objects import Vertex
7 from src.simplify.path import Path, prev_map,
disc_map, BFSQueue
8 from src.simplify.weighted_digraph import
WeightedDigraph
9
10
11 class TestPath(TestCase):
12     """Tests searching for BFS"""
13
14     def setUp(self):
15         # make a graph with 5 nodes
16
17         labels = ["a", "b", "c", "d", "e", "f"]
18         self.vertices = [Vertex(ID, label=label) for
ID, label in enumerate(labels)]
19
20         self.g = Digraph(self.vertices)
21         a, b, c, d, e, f = self.vertices
22         self.g.add_edge(a, b, c)
23         self.g.add_edge(b, d)
24         self.g.add_edge(d, f)
25         self.g.add_edge(c, e)
26         self.g.add_edge(d, f)
27         self.g.add_edge(e, f, b)
28
29         # set up weighted_graph and flow graphs
30         self.weighted_graph = WeightedDigraph(self.
vertices)
31         self.flow_graph = FlowGraph(self.vertices)
32
33         for g in [self.weighted_graph, self.
flow_graph]:
34             g.add_edge(a, (b, 10), (c, 10))
35             g.add_edge(b, (d, 25))
36             g.add_edge(c, (e, 25))
```

```

37             g.add_edge(d, (f, 10))
38             g.add_edge(e, (f, 10), (b, 6))
39
40     def test_shortest_path(self):
41         """
42             Checks that we can in fact find shortest
43             path along
44             """
45
46             a, b, c, d, e, f = self.vertices
47
48             cases = ["digraph", "weighted_graph", "flow"]
49
50             # check precalculated shortest path
51             graph_cases = [self.g, self.weighted_graph,
52                             self.flow_graph]
53
54             for case, g in zip(cases, graph_cases):
55                 with self.subTest(case):
56                     calc_shorted: list[Vertex] = Path.
57                     shortest_path(
58                         self.g, a, f, self.g.neighbours
59                     )
60                     expected: list[Vertex] = [a, c, e, f
61 ]
62                     self.assertEqual(expected,
63                     calc_shorted)
64
65             def test_build_path(self):
66                 """Given a prev_map, check we build the
67                 right path"""
68
69                 # generate vertices, unpack into vars
70                 vertices = []
71                 for ID, name in enumerate(["A", "B", "C", "D",
72                     "E", "F"]):
73                     vertices.append(Vertex(ID, name))
74
75                 a, c, b, d, e, f = vertices

```

```

70          # build a prev map. for context, path is A
71          # -> B -> D -> F
72          prev: prev_map = {a: None, b: a, c: a, d: b
73          , e: c, f: d}
74          expected = [a, b, d, f]
75
76      def test_recursive_bfs(self):
77          """Check that BFS is finding paths along
78          graph correctly"""
79          graph = self.flow_graph
80          a, b, c, d, e, f = graph.nodes()
81
82          expected: prev_map = {a: None, b: a, c: a,
83          d: b, e: c, f: e}
84
85          # set up structures for BFS
86          # create queue, discovered list, previous
87          # list
88          queue = BFSQueue(next(iter(graph.graph)))
89
90          discovered: disc_map = {node: False for
91          node in graph.nodes()}
92          prev: prev_map = {node: None for node in
93          graph.nodes()}
94
95          self.assertEqual(
96              expected,
97              Path.BFS(
98                  graph=graph,
99                  queue=queue,
100                 discovered=discovered,
101                 target=None,
102                 previous=prev,
103                 neighbours=graph.neighbours,
104                 ),
105             )
106
107      def test_find_target(self):

```

```
103     """Checks that BFS can stop when its given
104     target is found"""
105     graph = self.flow_graph
106     a, b, c, *_ = graph.nodes()
107     expected = {node: None for node in graph.
108     nodes()}
109     expected[b] = a
110     expected[c] = a
111     queue, discovered, previous = Path.
112     build_bfs_structs(graph, a)
113     calculated = Path.BFS(
114         graph=graph,
115         queue=queue,
116         discovered=discovered,
117         previous=previous,
118         target=b,
119         neighbours=graph.neighbours,
120     )
121     self.assertEqual(expected, calculated)
122
123     def test_build_bfs_struct(self):
124         """Checks that structures that are built
125         for BFS are built correctly"""
126         with self.subTest("with initial value"):
127             queue, disc, prev = Path.
128             build_bfs_structs(
129                 self.flow_graph, self.vertices[0]
130             )
131             self.assertEqual(queue, BFSQueue(self.
132             vertices[0]))
133         with self.subTest("with initial value"):
134             queue, disc, prev = Path.
135             build_bfs_structs(self.flow_graph)
136             self.assertEqual(queue, BFSQueue())
```

```

1 # coding=utf-8
2 from unittest import TestCase
3
4 from src.simplify.flow_algorithms import
5     NoOptimisations, MaxFlow, Simplify
6 from src.simplify.flow_graph import *
7 from src.simplify.graph_objects import Vertex
8
9
9 class TestFlowEdge(TestCase):
10     """Tests for FlowEdge"""
11
12     def setUp(self) -> None:
13         self.edges = [FlowEdge(Vertex(0), Vertex(1),
14             5 * n) for n in range(2)]
14         self.edges[0].flow = -3 # => unused
15         capacity should be three
15
16     def test_unused_capacity(self):
17         """builds two edges, residual and non-
18         residual
19             non-residual has capacity 5"""
20
20         for edge, cap in zip(self.edges, [3, 5]):
21             with self.subTest(edge):
22                 self.assertEqual(edge.
22 unused_capacity(), cap)
23
24     def test_push_flow(self):
25         """Checks that we update flow by required
26         amount"""
26         # push three units of capacity through each
27         edge
27         for edge, exp in zip(self.edges, [0, 2]):
28             with self.subTest(edge):
29                 edge.push_flow(3)
30                 self.assertEqual(exp, edge.
30 unused_capacity())
31
32         # try to push excess amount of flow down an
32         edge

```

```

33         with self.subTest("exceed capacity"), self.
34             assertRaises(FlowEdgeError):
35                 self.edges[1].push_flow(3)
36
37     def test_adjust_edge(self):
38         """Tests that the adjust_edges function
39         works as expected
40             i.e when adjust is called, edges in the
41             graph morph to edges with a flow equal to their
42             unused capacity
43             and all edges of weight 0 are deleted"""
44
45         (
46             res,
47             fwd,
48         ) = self.edges
49         fwd.push_flow(3)
50         res.push_flow(-3)
51
52         new_fwd = FlowEdge(Vertex(0), Vertex(1), 2)
53         new_res = FlowEdge(Vertex(0), Vertex(1), 0)
54
55         fwd.adjust_edge()
56         res.adjust_edge()
57
58         self.assertEqual(new_fwd, fwd)
59         self.assertEqual(new_res, res)
60
61
62     class TestFlowGraph(TestCase):
63         def setUp(self) -> None:
64             # make some nodes for a graph
65             self.nodes = [Vertex(n, label=chr(n + 97))
66             for n in range(5)]
67             self.graph = FlowGraph(self.nodes)
68
69         def test_is_edge(self):

```

File - /home/tcassar/projects/settle/tests/test_settling/test_flow.py

```
69     a, b, c, d, e = self.nodes
70
71     with self.subTest("no edge"):
72         self.assertFalse(self.graph.is_edge(a,
73                           b))
74
75     with self.subTest("edge"):
76         self.graph.add_edge(a, (b, 4))
77         self.assertTrue(self.graph.is_edge(a, b))
78
79     def test_get_edge(self):
80         a, b, c, d, e = self.graph.nodes()
81         self.graph.add_edge(a, (b, 5))
82         self.assertEqual(FlowEdge(a, b, 5), self.
83             graph.get_edge(a, b))
84
85     def test_add_edge(self):
86         a, b, c, d, e = self.nodes
87         # check there aren't edges
88         with self.subTest("negative test"):
89             self.assertFalse(self.graph.is_edge(a,
90                           b))
91             self.assertFalse(self.graph.is_edge(b,
92                           a, residual=True))
93             self.graph.to_dot()
94             self.graph.add_edge(a, (b, 5))
95             self.graph.to_dot()
96             with self.subTest("added"):
97                 self.assertTrue(self.graph.is_edge(a, b))
98                 self.assertTrue(self.graph.is_edge(b, a,
99                               , residual=True))
100                self.assertFalse(self.graph.is_edge(b,
101                               a))
102
103    with self.subTest("Net debt (adding)"):
104        self.assertEqual(
105            {
106                Vertex(ID=0, label="a"): 5,
107                Vertex(ID=1, label="b"): -5,
```

```
102                     Vertex(ID=2, label="c"): 0,
103                     Vertex(ID=3, label="d"): 0,
104                     Vertex(ID=4, label="e"): 0,
105                 },
106                 self.graph.net_debt,
107             )
108
109         with self.subTest("Net debt (removing)"):
110             self.graph.pop_edge(a, b, update_debt=
True)
111             self.assertEqual(
112                 {
113                     Vertex(ID=0, label="a"): 0,
114                     Vertex(ID=1, label="b"): 0,
115                     Vertex(ID=2, label="c"): 0,
116                     Vertex(ID=3, label="d"): 0,
117                     Vertex(ID=4, label="e"): 0,
118                 },
119                 self.graph.net_debt,
120             )
121
122     def test_pop_edge(self):
123         a, b, c, d, e = self.nodes
124
125         self.graph.add_edge(a, (b, 5))
126
127         with self.subTest("negative"):
128             self.assertTrue(self.graph.is_edge(a, b))
129             self.assertTrue(self.graph.is_edge(b, a,
, residual=True))
130
131         self.graph.pop_edge(a, b)
132
133         with self.subTest("removed edge"):
134             self.assertFalse(self.graph.is_edge(a,
b))
135         with self.subTest("removed residual"):
136             self.assertFalse(self.graph.is_edge(b,
a, residual=True))
137
```

```

138     def test_flow_neighbours(self):
139         """Checks we get edges that have unused
140             capacity, including residual
141             These edges count as valid paths to
142             neighbouring nodes in Edmonds-Karp"""
143         graph = self.graph
144         a, b, c, d, *_ = graph.nodes()
145         graph.add_edge(a, (b, 10))
146         graph.add_edge(b, (c, 2))
147         graph.add_edge(c, (d, 10))
148         graph.add_edge(d, (a, 10))
149
150         MaxFlow.augment_flow(graph, [a, b, c, d], 2
151     )
152         c_flow_neighbours = graph.flow_neighbours(c
153     )
154         self.assertEqual([d, b], GenericDigraph.
155             nodes_from_edges(c_flow_neighbours))
156
157     def test_bool(self):
158         """Tests that dunder bool method returns
159             true when there are edges in the graph"""
160         # empty
161         print(self.graph.graph)
162         self.assertFalse(bool(self.graph))
163
164         # with an edge
165         a, b, c, d, e = self.nodes
166         self.graph.add_edge(a, (b, 10))
167
168         def test_add_existing(self):
169             """Tests that adding an existing edge will
170                 add capacity onto the existing edge,
171                 as opposed to adding a new distinct edge
172                 from src -> dest"""
173             a, b, c, d, e = self.graph.nodes()
174             self.graph.add_edge(a, (b, 5))
175             self.graph.add_edge(a, (b, 5))
176             print(self.graph)

```

```

171         self.assertEqual(self.graph.get_edge(a, b).
172                         capacity, 10)
173
174     def test_pop_existing(self):
175         """Tests that if an edge A -> B of weight [5] exists, adding B -> A [5] will remove
176             any edges between A and B, as there is net
177             0 between the two"""
178
179         a, b, c, d, e = self.graph.nodes()
180         self.graph.add_edge(a, (b, 5))
181         self.graph.add_edge(b, (a, 5))
182
183         self.assertFalse(self.graph.is_edge(a, b))
184
185     def test_add_edge_reverses_new_edge(self):
186         """Checks that if there exists an edge A
187             -> B of weight [5] exists, adding B -> A [10] will
188             reverse the direction of the edge such that
189             now an edge B -> A [5] exists"""
190
191         a, b, c, d, e = self.graph.nodes()
192         self.graph.add_edge(a, (b, 5))
193         self.graph.add_edge(b, (a, 10))
194
195         self.assertFalse(self.graph.is_edge(a, b))
196         self.assertEqual(self.graph.get_edge(b, a).
197                         capacity, 5)
198
199
200 class TestMaxFlow(TestCase):
201     def setUp(self) -> None:
202         graph = FlowGraph([Vertex(n, label=chr(n +
203                                     97)) for n in range(4)])
204
205         a, b, c, d, *_ = graph.nodes()
206         graph.add_edge(a, (b, 10))
207         graph.add_edge(b, (c, 2))
208         graph.add_edge(c, (d, 10))
209         graph.add_edge(d, (a, 10))
210
211         self.graph = graph
212
213     def test_bottleneck(self):

```

```

205         """Checks we get the correct bottleneck
206         value on a path"""
207         # build a chain with an obvious bottleneck
208         # on normal graphs
209         graph = self.graph
210         a, b, c, d, *_ = graph.nodes()
211
212         aug_path = [a, b, c, d]
213
214         graph.to_dot()
215
216     def test_nodes_to_path(self):
217         """Checks that we can build a path of edges
218         from nodes"""
219         a, b, c, d, *_ = self.graph.nodes()
220         edges = MaxFlow.nodes_to_path(self.graph, [
221             a, b, c, d])
222
223         self.assertEqual(
224             edges, [FlowEdge(a, b, 10), FlowEdge(b,
225                 c, 2), FlowEdge(c, d, 10)])
226
227     def test_augmenting_path(self):
228         """Tests that we find valid augmenting
229         paths"""
230         a, b, c, d, *_ = self.graph.nodes()
231         MaxFlow.augmenting_path(self.graph, a, d)
232
233     def test_augment_flow(self):
234         """Checks that augmenting flow works
235         exactly as described in Analysis"""
236         a, b, c, d, *_ = self.graph.nodes()
237
238         MaxFlow.augment_flow(self.graph, [a, b, c,
239             d], 2)
240
241         self.graph.to_dot()
242
243

```

File - /home/tcassar/projects/settle/tests/test_settling/test_flow.py

```

237         # check that the edges and residual edges
238         all now have flow 2
239         node_path = [a, b, c, d]
240         flow = 2
241         for _ in range(2):
242             # one for normal one for residual
243             for src, dest in zip(node_path,
244             node_path[1:]):
245                 with self.subTest(f"{src} -> {dest}"):
246                     self.assertEqual(flow, self.
247                         graph.get_edge(src, dest).flow)
248                     # change path to be residual, flow
249                     changes accordingly
250                     node_path.reverse()
251                     flow *= -1
252
253     def test_edmonds_karp(self):
254         """Integration test for edmonds-karp,
255         ensures that max flow is correct"""
256         # build slightly more involved graph
257         nodes = [Vertex(0, label="src"), Vertex(10
258         , label="sink")]
259         nodes += [Vertex(n, label=chr(n + 96)) for
260         n in range(1, 10)]
261         tg = FlowGraph(nodes)
262
263         s, t, a, b, c, d, e, f, g, h, i = tg.nodes
264         ()
265
266         tg.add_edge(s, (a, 5), (b, 10), (c, 5))
267         tg.add_edge(a, (d, 10))
268         tg.add_edge(b, (a, 15), (e, 20))
269         tg.add_edge(c, (f, 10))
270         tg.add_edge(d, (e, 25), (g, 10))
271         tg.add_edge(e, (c, 5), (h, 30))
272         tg.add_edge(f, (h, 5), (i, 5))
273         tg.add_edge(g, (t, 5))
274         tg.add_edge(h, (t, 15), (i, 5))
275         tg.add_edge(i, (t, 10))
276
277

```

```

269         tg.to_dot()
270
271         max_flow = MaxFlow.edmonds_karp(tg, s, t)
272         self.assertEqual(max_flow, 20)
273
274     def test_old_edmonds(self):
275         """Tests a more complex graph for correct
maxflow"""
276         labels = ["a", "b", "c", "d", "e", "f"]
277         self.vertices = [Vertex(ID, label=label)
278             for ID, label in enumerate(labels)]
279         a, b, c, d, e, f = self.vertices
280
281         # set up weighted_graph and flow graphs
282         self.flow_graph = FlowGraph(self.vertices)
283         self.flow_graph.add_edge(a, (b, 10), (c, 10))
284         self.flow_graph.add_edge(b, (d, 25))
285         self.flow_graph.add_edge(c, (e, 25))
286         self.flow_graph.add_edge(d, (f, 10))
287         self.flow_graph.add_edge(e, (f, 10), (b, 6))
288         a, b, c, d, e, f = self.vertices
289
290         expected = 20
291         calculated = MaxFlow.edmonds_karp(self.
292             flow_graph, a, f)
293
294         self.assertEqual(expected, calculated)
295
296 class TestSimplify(TestCase): # type: ignore
297     def setUp(self) -> None:
298         self.graph = FlowGraph([Vertex(0, "d"),
299             Vertex(1, "m"), Vertex(2, "t")])
300         d, m, t = self.graph.nodes()
301         self.graph.add_edge(d, (m, 5), (t, 10))
302         self.graph.add_edge(m, (t, 5))
303
304     def test_simplify_debt(self):

```

```

304         """Uses graph from analysis to test that
305             the debt simplifying algorithm simplifies debt,
306             does not create or
307                 destroy debt"""
308
309     print(self.graph.net_debt)
310
311     people = ["dad", "tom", "maia"]
312     debt = FlowGraph([Vertex(ID, person) for ID
313 , person in enumerate(people)])
314
315     d, t, m = debt.nodes()
316
317     debt.add_edge(d, (t, 10), (m, 5))
318     debt.add_edge(m, (t, 5))
319
320     debt.to_dot()
321
322     clean = Simplify.simplify_debt(debt)
323
324     def test_adjust_edges(self):
325
326         """Checks that edges are adjusted
327             accordingly, generate graphs before and after"""
328
329         # saturate d -> m -> t
330         self.graph.to_dot(n=0)
331         d, m, t = self.graph.nodes()
332         MaxFlow.augment_flow(self.graph, [d, m, t
333 ], 5)
334         self.graph.to_dot(n=1)
335         self.graph.adjust_edges()
336         self.graph.to_dot(n=4)
337
338         # graph should have no edges in or out of m
339         # t should have a res edge to d, 0/0
340         # d should have a fwd to t, 0/10

```

```

339
340         self.assertFalse(self.graph[m])
341         self.assertEqual(self.graph[d], [FlowEdge(d
342             , t, 10)])
343         self.assertEqual(self.graph[t], [FlowEdge(t
344             , d, 0)])
345
346     def test_mithun_simplify(self):
347
348         """Checks more complex example, ensures
349         that graph is settled and that no debt is created
350         / destroyed"""
351
352         # gen vertices
353         people: list[Vertex] = []
354         for ID, person in enumerate(["b", "c", "d"
355             , "e", "f", "g"]):
356             people.append(Vertex(ID, label=person))
357         b, c, d, e, f, g = people
358
359         # build flow graph of transactions
360         messy = FlowGraph(people)
361
362         messy.add_edge(b, (c, 40))
363         messy.add_edge(c, (d, 20))
364         messy.add_edge(d, (e, 50))
365         messy.add_edge(f, (e, 10), (d, 10), (c, 30
366             ), (b, 10))
367         messy.add_edge(g, (b, 30), (d, 10))
368
369         messy.to_dot()
370
371         clean = Simplify.simplify_debt(messy)
372         clean.to_dot(n=1)
373
374         self.assertEqual(messy.net_debt, clean.
375             net_debt)
376
377     def test_simplest_graph(self):
378         """Shouldn't change"""
379         people = ["dad", "tom", "maia"]

```

```
373     debt = FlowGraph([Vertex(ID, person) for ID
374         , person in enumerate(people)])
375         d, t, m = debt.nodes()
376         debt.add_edge(d, (t, 5))
377         debt.add_edge(t, (m, 10))
378
379         debt.to_dot()
380
381     with self.assertRaises(NoOptimisations):
382         clean = Simplify.simplify_debt(debt)
383         clean.to_dot(n=1)
```

```
1 # coding=utf-8
2
3 from unittest import TestCase
4
5 from src.simplify.base_graph import Digraph
6 from src.simplify.flow_graph import *
7 from src.simplify.graph_objects import Edge
8 from src.simplify.weighted_digraph import
9     WeightedDigraph
10
11 class TestDigraph(TestCase):
12     """Generate a digraph and some vertices"""
13
14     def setUp(self) -> None:
15         """Build basic graph"""
16         labels = ["u", "v", "w"]
17         self.vertices = [Vertex(ID, label=label) for
18             ID, label in enumerate(labels)]
19
20         self.graph = Digraph(self.vertices)
21         u, v, w = self.vertices
22         self.graph.add_edge(u, v, w)
23         self.graph.add_edge(v, w)
24
25     def test_init(self):
26         expected = "U -> V\nV -> W\nW -> \n"
27
28         with self.subTest("init"):
29             self.assertEqual(expected, str(self.
graph))
30
31         with self.subTest("helpers"):
32             self.assertTrue(self.graph.is_node(self.
vertices[0]))
33             self.assertTrue(self.graph.sanitize(*
self.vertices))
34
35             with self.assertRaises(GraphError):
36                 self.graph.edge_from_nodes(self.
vertices[2], [Edge(self.vertices[0])])
```

```
36
37     def test_is_edge(self):
38         u, v, w = self.vertices
39
40         with self.subTest("is edge"):
41             self.assertTrue(self.graph.is_edge(u, v))
42         with self.subTest("isn't edge"):
43             self.assertFalse(self.graph.is_edge(w, v))
44
45     def test_add_node(self):
46
47         new = Vertex(4, "b")
48         self.assertFalse(self.graph.is_node(new))
49
50         self.graph.add_node(new)
51         self.assertTrue(self.graph.is_node(new))
52
53     def test_pop_node(self):
54         u, v, w = self.vertices
55
56         print(self.graph)
57         self.graph.pop_node(u)
58         print(self.graph)
59         self.assertFalse(self.graph.is_node(u))
60         with self.assertRaises(GraphError):
61             self.graph.edge_from_nodes(u, self.graph
62 [v])
63
64     def test_pop_edge(self):
65         u, v, w = self.vertices
66         self.assertTrue(self.graph.is_edge(u, v))
67         self.graph.pop_edge(u, v)
68
69         self.assertFalse(self.graph.is_edge(u, v))
70
71     def test_add_edge(self):
72         u, v, w = self.graph.nodes()
73         self.assertFalse(self.graph.is_edge(w, v))
74         self.graph.add_edge(w, v)
```

```
74
75             self.assertTrue(self.graph.is_edge(w, v))
76
77     def test_nodes(self):
78         self.assertEqual(self.vertices, self.graph.
nodes())
79
80
81 class TestWeightedDigraph(TestCase):
82     """Artefact of prototyping process"""
83     def setUp(self) -> None:
84         """Build basic graph"""
85         labels = ["u", "v", "w"]
86         self.vertices = [Vertex(ID, label=label)
87             for ID, label in enumerate(labels)]
88
89         self.graph = WeightedDigraph(self.vertices)
90         u, v, w = self.vertices
91         self.graph.add_edge(u, (v, 1), (w, 2))
92         self.graph.add_edge(v, (w, 3))
93
94     def test_add_edge(self):
95         u, v, w = self.vertices
96         self.assertFalse(self.graph.is_edge(w, v))
97         self.graph.add_edge(w, (v, 4))
98         self.assertTrue(self.graph.is_edge(w, v))
99
100    def test_add_existing_edge(self):
101        u, v, w = self.graph.nodes()
102        self.assertEqual(self.graph.is_edge(v, w),
103            3)
104        self.graph.add_edge(v, (w, 2))
105        self.assertEqual(self.graph.is_edge(v, w),
106            5)
107
108    def test_flow_through(self):
109        for node, flow in zip(self.vertices, [3, 2,
110            -5]):
111            with self.subTest(node):
112                self.assertEqual(self.graph.
flow_through(node), flow)
```


ID	src	dest	amount	group	src_n	src_e	dest_n	dest_e	src_d	dest_d
2	0,04,20,5,0,	2161679203114375274630941557945232020251089312607308 7652383723408105063600070147761500936454570470862786 9702069833686361660030089751732511247633740543213460 3035445702142516535603778163693003610640441829541372 643100255683690006704986794499904312842155745102543 7279819137427553642625019821057158620227234339857381 3906769437847646651445567701079686709061678948557745 8637370869261774337704654931841775536224420241750390 3181827421441408785602795322818434551136750209831842 5250366913849245188288359620277590091100050269612943 3770297061845035322423193568722584809589006962060587 421954603201784497846158263582144177430642603,65537, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 7308477851944847288700266021218281412607086300444035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021,65537, 4231860502610347555352698504423047563944256813822996 3925734038989576569874864579669508371562955146126547 8779552919144303233011283933461726857674426838187389 6721663847672076407494464781754007098969569658821095 1082558372250397158894945714319204774976709066583095 9656389449195952398626578010004019881519663179628663 1049344644422180019003905454847914081363056944472084 7689992964879750244948034517871603890465032993904165 5974782350016040380747036826804133887037575013624446 8012222270131756222483185448064466203435454227528122 7314387046825525570376824278684460211954070416261823 5709118874694072838664235842554512315661905, 1745164279842191683897870688775252929437856379494395 9644593481018317996812806135240585387347011403463919								

2	4843123602886715197650891859619406775380829934052483 5872485281606266439620861446034635607786292042673482 7642392974806673957359914994399933521690425914379137 2656087371539940176580242716026174273279118970901055 4280600760265229078672123452862613189683248956255155 8690301697229495263298457688434655931885453437842161 1289899532022728673678378918475624105043819026629471 6820800459652950177227887446325164917035367452040210 6924748077589683971300268994119580437205561936946700 600893560778543423518093933835790069197852673
3	1, 04, 13, 10, 0, 2161679203114375274630941557945232020251089312607308 7652383723408105063600070147761500936454570470862786 9702069833686361660030089751732511247633740543213460 3035445702142516535603778163693003610640441829541372 643100255683690006704986794499904312842155745102543 7279819137427553642625019821057158620227234339857381 3906769437847646651445567701079686709061678948557745 8637370869261774337704654931841775536224420241750390 3181827421441408785602795322818434551136750209831842 5250366913849245188288359620277590091100050269612943 3770297061845035322423193568722584809589006962060587 421954603201784497846158263582144177430642603, 65537, 2256586403724745983208556687990030736021636869740873 3883964851808410874820858287921659997954743444212625 8889014725245633691567258383521676090489827618604882 3446599370109121575718012944124229461713591222435833 0826328284855986303335327620511972006453142090664344 6610551234084941258572108217677529623697862636846419 0409612760214589978439512179632414787894743946995271 6058077163515372836183503361380147895905992145482259 1898567781991786261932874106056739293869626830428217 3136210695524457478834244215547988454805806522190171 0928067881317959628155388903833923070328443328378726 386495866615509251768955425927155295624588823, 65537, 4231860502610347555352698504423047563944256813822996 3925734038989576569874864579669508371562955146126547 8779552919144303233011283933461726857674426838187389 6721663847672076407494464781754007098969569658821095 1082558372250397158894945714319204774976709066583095

3	9656389449195952398626578010004019881519663179628663 104934464422180019003905454847914081363056944472084 7689992964879750244948034517871603890465032993904165 5974782350016040380747036826804133887037575013624446 8012222270131756222483185448064466203435454227528122 7314387046825525570376824278684460211954070416261823 5709118874694072838664235842554512315661905, 7754142821899275148672764486249827147291951463534258 3131191306090515724089556837205820094594019006617381 7870914386153115146267705082153309665964404530415177 7471873569870060864117925980999686814744354231846889 6990846846049836207197701758385280007526246224538969 0963824067563800165256859035674194291114312004167623 5694762226761908445600472087203109098947447060087682 640572923907398373569449250780454435298169779944004 8112763789825376382626011524853438555391048758769078 0789232355207322596895153032599004522731021331970408 1395654861368829855439175850787537877943284782407352 5835250217870849030986866114641442612783873
4	2,20,13,5,0, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 730847785194484728870026602121828141260708630044035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021,65537, 2256586403724745983208556687990030736021636869740873 3883964851808410874820858287921659997954743444212625 8889014725245633691567258383521676090489827618604882 3446599370109121575718012944124229461713591222435833 0826328284855986303335327620511972006453142090664344 6610551234084941258572108217677529623697862636846419 0409612760214589978439512179632414787894743946995271 6058077163515372836183503361380147895905992145482259

4

1898567781991786261932874106056739293869626830428217
313621069552445747834244215547988454805806522190171
0928067881317959628155388903833923070328443328378726
386495866615509251768955425927155295624588823, 65537,
1745164279842191683897870688775252929437856379494395
9644593481018317996812806135240585387347011403463919
4843123602886715197650891859619406775380829934052483
5872485281606266439620861446034635607786292042673482
7642392974806673957359914994399933521690425914379137
2656087371539940176580242716026174273279118970901055
4280600760265229078672123452862613189683248956255155
8690301697229495263298457688434655931885453437842161
1289899532022728673678378918475624105043819026629471
6820800459652950177227887446325164917035367452040210
6924748077589683971300268994119580437205561936946700
600893560778543423518093933835790069197852673,
7754142821899275148672764486249827147291951463534258
3131191306090515724089556837205820094594019006617381
7870914386153115146267705082153309665964404530415177
7471873569870060864117925980999686814744354231846889
699084684604983620719770175838528007526246224538969
0963824067563800165256859035674194291114312004167623
5694762226761908445600472087203109098947447060087682
6405729239073983735694492507804544435298169779944004
8112763789825376382626011524853438555391048758769078
0789232355207322596895153032599004522731021331970408
1395654861368829855439175850787537877943284782407352
5835250217870849030986866114641442612783873

5 3, 04, 20, 5, 1,

2161679203114375274630941557945232020251089312607308
7652383723408105063600070147761500936454570470862786
9702069833686361660030089751732511247633740543213460
3035445702142516535603778163693003610640441829541372
643100255683690006704986794499904312842155745102543
7279819137427553642625019821057158620227234339857381
3906769437847646651445567701079686709061678948557745
8637370869261774337704654931841775536224420241750390
3181827421441408785602795322818434551136750209831842
5250366913849245188288359620277590091100050269612943
3770297061845035322423193568722584809589006962060587

5	421954603201784497846158263582144177430642603, 65537, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 7308477851944847288700266021218281412607086300444035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021, 65537, 4231860502610347555352698504423047563944256813822996 3925734038989576569874864579669508371562955146126547 8779552919144303233011283933461726857674426838187389 6721663847672076407494464781754007098969569658821095 1082558372250397158894945714319204774976709066583095 9656389449195952398626578010004019881519663179628663 104934464422180019003905454847914081363056944472084 7689992964879750244948034517871603890465032993904165 5974782350016040380747036826804133887037575013624446 8012222270131756222483185448064466203435454227528122 7314387046825525570376824278684460211954070416261823 570911887469407283864235842554512315661905, 1745164279842191683897870688775252929437856379494395 9644593481018317996812806135240585387347011403463919 4843123602886715197650891859619406775380829934052483 5872485281606266439620861446034635607786292042673482 7642392974806673957359914994399933521690425914379137 2656087371539940176580242716026174273279118970901055 4280600760265229078672123452862613189683248956255155 8690301697229495263298457688434655931885453437842161 1289899532022728673678378918475624105043819026629471 6820800459652950177227887446325164917035367452040210 6924748077589683971300268994119580437205561936946700 600893560778543423518093933835790069197852673
6	4, 04, 13, 10, 1, 2161679203114375274630941557945232020251089312607308 7652383723408105063600070147761500936454570470862786 9702069833686361660030089751732511247633740543213460

6

3035445702142516535603778163693003610640441829541372
6431002556836900067049867944499904312842155745102543
7279819137427553642625019821057158620227234339857381
3906769437847646651445567701079686709061678948557745
8637370869261774337704654931841775536224420241750390
3181827421441408785602795322818434551136750209831842
5250366913849245188288359620277590091100050269612943
3770297061845035322423193568722584809589006962060587
421954603201784497846158263582144177430642603, 65537,
2256586403724745983208556687990030736021636869740873
388396485180841087482085828792165999795474344212625
8889014725245633691567258383521676090489827618604882
3446599370109121575718012944124229461713591222435833
0826328284855986303335327620511972006453142090664344
6610551234084941258572108217677529623697862636846419
0409612760214589978439512179632414787894743946995271
6058077163515372836183503361380147895905992145482259
1898567781991786261932874106056739293869626830428217
313621069552445747834244215547988454805806522190171
0928067881317959628155388903833923070328443328378726
38649586661550925176895542592715529562458823, 65537,
4231860502610347555352698504423047563944256813822996
3925734038989576569874864579669508371562955146126547
8779552919144303233011283933461726857674426838187389
6721663847672076407494464781754007098969569658821095
1082558372250397158894945714319204774976709066583095
9656389449195952398626578010004019881519663179628663
104934464422180019003905454847914081363056944472084
7689992964879750244948034517871603890465032993904165
5974782350016040380747036826804133887037575013624446
8012222270131756222483185448064466203435454227528122
7314387046825525570376824278684460211954070416261823
5709118874694072838664235842554512315661905,
7754142821899275148672764486249827147291951463534258
3131191306090515724089556837205820094594019006617381
7870914386153115146267705082153309665964404530415177
7471873569870060864117925980999686814744354231846889
6990846846049836207197701758385280007526246224538969
096382406756380016525685903567419429114312004167623
5694762226761908445600472087203109098947447060087682

6	640572923907398373569449250780454435298169779944004 8112763789825376382626011524853438555391048758769078 0789232355207322596895153032599004522731021331970408 1395654861368829855439175850787537877943284782407352 5835250217870849030986866114641442612783873
7	5, 20, 13, 5, 1, 2454352605322268592008900243138707108080875397916828 9339650707414109582770834242173009539282426745682722 9880553467004005448679109437851439095238915265836858 7308477851944847288700266021218281412607086300444035 1914152540555901097499930235793743416545610368040032 5960557898468091402415007872965823676993425321804474 4604285155918762905223816899981984372615636383003006 9139123051200441111945407276884741033640730462395861 9851439941769916099206947465717047070945326989108443 5772741978366746154234093028750651579356970414938478 4131969168222943901917462927710491717935881329934336 642631167278369058626398091411958822782486021, 65537, 2256586403724745983208556687990030736021636869740873 3883964851808410874820858287921659997954743444212625 8889014725245633691567258383521676090489827618604882 3446599370109121575718012944124229461713591222435833 0826328284855986303335327620511972006453142090664344 6610551234084941258572108217677529623697862636846419 0409612760214589978439512179632414787894743946995271 6058077163515372836183503361380147895905992145482259 1898567781991786261932874106056739293869626830428217 3136210695524457478834244215547988454805806522190171 0928067881317959628155388903833923070328443328378726 386495866615509251768955425927155295624588823, 65537, 1745164279842191683897870688775252929437856379494395 9644593481018317996812806135240585387347011403463919 4843123602886715197650891859619406775380829934052483 5872485281606266439620861446034635607786292042673482 7642392974806673957359914994399933521690425914379137 2656087371539940176580242716026174273279118970901055 4280600760265229078672123452862613189683248956255155 8690301697229495263298457688434655931885453437842161 1289899532022728673678378918475624105043819026629471 6820800459652950177227887446325164917035367452040210

7

6924748077589683971300268994119580437205561936946700
600893560778543423518093933835790069197852673,
7754142821899275148672764486249827147291951463534258
3131191306090515724089556837205820094594019006617381
7870914386153115146267705082153309665964404530415177
7471873569870060864117925980999686814744354231846889
6990846846049836207197701758385280007526246224538969
0963824067563800165256859035674194291114312004167623
5694762226761908445600472087203109098947447060087682
640572923907398373569449250780454435298169779944004
8112763789825376382626011524853438555391048758769078
0789232355207322596895153032599004522731021331970408
1395654861368829855439175850787537877943284782407352
5835250217870849030986866114641442612783873

8 6,04,20,5,2,

2161679203114375274630941557945232020251089312607308
7652383723408105063600070147761500936454570470862786
9702069833686361660030089751732511247633740543213460
3035445702142516535603778163693003610640441829541372
6431002556836900067049867944499904312842155745102543
7279819137427553642625019821057158620227234339857381
3906769437847646651445567701079686709061678948557745
8637370869261774337704654931841775536224420241750390
3181827421441408785602795322818434551136750209831842
5250366913849245188288359620277590091100050269612943
3770297061845035322423193568722584809589006962060587
421954603201784497846158263582144177430642603,65537,
2454352605322268592008900243138707108080875397916828
9339650707414109582770834242173009539282426745682722
9880553467004005448679109437851439095238915265836858
7308477851944847288700266021218281412607086300444035
1914152540555901097499930235793743416545610368040032
5960557898468091402415007872965823676993425321804474
4604285155918762905223816899981984372615636383003006
9139123051200441111945407276884741033640730462395861
9851439941769916099206947465717047070945326989108443
5772741978366746154234093028750651579356970414938478
4131969168222943901917462927710491717935881329934336
642631167278369058626398091411958822782486021,65537,
4231860502610347555352698504423047563944256813822996

8	3925734038989576569874864579669508371562955146126547 8779552919144303233011283933461726857674426838187389 6721663847672076407494464781754007098969569658821095 1082558372250397158894945714319204774976709066583095 9656389449195952398626578010004019881519663179628663 1049344644422180019003905454847914081363056944472084 7689992964879750244948034517871603890465032993904165 5974782350016040380747036826804133887037575013624446 8012222270131756222483185448064466203435454227528122 7314387046825525570376824278684460211954070416261823 5709118874694072838664235842554512315661905, 1745164279842191683897870688775252929437856379494395 9644593481018317996812806135240585387347011403463919 4843123602886715197650891859619406775380829934052483 5872485281606266439620861446034635607786292042673482 7642392974806673957359914994399933521690425914379137 2656087371539940176580242716026174273279118970901055 4280600760265229078672123452862613189683248956255155 8690301697229495263298457688434655931885453437842161 1289899532022728673678378918475624105043819026629471 6820800459652950177227887446325164917035367452040210 6924748077589683971300268994119580437205561936946700 600893560778543423518093933835790069197852673
9	7, 04, 13, 10, 2, 65537, 6216167920311437527463094155794523202025108931260730 8765238372340810506360007014776150093645457047086278 6970206983368636166003008975173251124763374054321346 0303544570214251653560377816369300361064044182954137 2643100255683690006704986794449990431284215574510254 3727981913742755364262501982105715862022723433985738 1390676943784764665144556770107968670906167894855774 5863737086926177433770465493184177553622442024175039 0318182742144140878560279532281843455113675020983184 2525036691384924518828835962027759009110005026961294 3377029706184503532242319356872258480958900696206058 74219546032017844978461582635821441774306426035537, 2256586403724745983208556687990030736021636869740873 3883964851808410874820858287921659997954743444212625 8889014725245633691567258383521676090489827618604882 3446599370109121575718012944124229461713591222435833

9

0826328284855986303335327620511972006453142090664344
6610551234084941258572108217677529623697862636846419
0409612760214589978439512179632414787894743946995271
6058077163515372836183503361380147895905992145482259
1898567781991786261932874106056739293869626830428217
313621069552445747834244215547988454805806522190171
0928067881317959628155388903833923070328443328378726
386495866615509251768955425927155295624588823, 65537,
4231860502610347555352698504423047563944256813822996
3925734038989576569874864579669508371562955146126547
8779552919144303233011283933461726857674426838187389
6721663847672076407494464781754007098969569658821095
1082558372250397158894945714319204774976709066583095
9656389449195952398626578010004019881519663179628663
1049344644422180019003905454847914081363056944472084
7689992964879750244948034517871603890465032993904165
5974782350016040380747036826804133887037575013624446
8012222270131756222483185448064466203435454227528122
7314387046825525570376824278684460211954070416261823
5709118874694072838664235842554512315661905,
7754142821899275148672764486249827147291951463534258
3131191306090515724089556837205820094594019006617381
7870914386153115146267705082153309665964404530415177
7471873569870060864117925980999686814744354231846889
6990846846049836207197701758385280007526246224538969
0963824067563800165256859035674194291114312004167623
5694762226761908445600472087203109098947447060087682
6405729239073983735694492507804544435298169779944004
8112763789825376382626011524853438555391048758769078
0789232355207322596895153032599004522731021331970408
1395654861368829855439175850787537877943284782407352
5835250217870849030986866114641442612783873

10 8, 20, 13, 5, 2,

2454352605322268592008900243138707108080875397916828
9339650707414109582770834242173009539282426745682722
9880553467004005448679109437851439095238915265836858
7308477851944847288700266021218281412607086300444035
1914152540555901097499930235793743416545610368040032
5960557898468091402415007872965823676993425321804474
4604285155918762905223816899981984372615636383003006

10

9139123051200441111945407276884741033640730462395861
9851439941769916099206947465717047070945326989108443
5772741978366746154234093028750651579356970414938478
4131969168222943901917462927710491717935881329934336
642631167278369058626398091411958822782486021, 65537,
2256586403724745983208556687990030736021636869740873
388396485180841087482085828792165999795474344212625
8889014725245633691567258383521676090489827618604882
3446599370109121575718012944124229461713591222435833
0826328284855986303335327620511972006453142090664344
6610551234084941258572108217677529623697862636846419
0409612760214589978439512179632414787894743946995271
6058077163515372836183503361380147895905992145482259
1898567781991786261932874106056739293869626830428217
3136210695524457478834244215547988454805806522190171
092806788131795962815538890383392307032843328378726
386495866615509251768955425927155295624588823, 65537,
1745164279842191683897870688775252929437856379494395
9644593481018317996812806135240585387347011403463919
4843123602886715197650891859619406775380829934052483
5872485281606266439620861446034635607786292042673482
7642392974806673957359914994399933521690425914379137
2656087371539940176580242716026174273279118970901055
4280600760265229078672123452862613189683248956255155
8690301697229495263298457688434655931885453437842161
1289899532022728673678378918475624105043819026629471
6820800459652950177227887446325164917035367452040210
6924748077589683971300268994119580437205561936946700
600893560778543423518093933835790069197852673,
7754142821899275148672764486249827147291951463534258
3131191306090515724089556837205820094594019006617381
7870914386153115146267705082153309665964404530415177
7471873569870060864117925980999686814744354231846889
6990846846049836207197701758385280007526246224538969
0963824067563800165256859035674194291114312004167623
5694762226761908445600472087203109098947447060087682
6405729239073983735694492507804544435298169779944004
81127637898253763826011524853438555391048758769078
0789232355207322596895153032599004522731021331970408
1395654861368829855439175850787537877943284782407352

10 5835250217870849030986866114641442612783873

11

```

1 # coding=utf-8
2 import unittest
3
4 import src.simplify.flow_graph
5 import src.simplify.graph_objects
6 from src.crypto import keys
7 from src.transactions.ledger import *
8 from src.transactions.transaction import
    VerificationError
9
10
11 def setUpModule():
12     print(os.getcwd())
13     os.chdir("/home/tcassar/projects/settle")
14
15
16 def key_path(usr: str, keytype="private") -> str:
17     assert usr == "d" or usr == "m" or usr == "t"
18     return f"./src/crypto/sample_keys/{usr}_{keytype}
    }-key.pem' if keytype == 'private' else ''}"
19
20
21 class TestLedger(unittest.TestCase):
22     def setUp(self) -> None:
23
        """Load in sample data from mock db"""
24
25
26     self.valid: Ledger
27     self.missing_key: Ledger
28     self.invalid: Ledger
29
30     # load in raw copies of d, m, t test keys
31     ldr = keys.RSAKeyLoader()
32     self.d_m_t_keys: dict[int, keys.
        RSAPrivateKey] = {}
33     pub: list[keys.RSAPublicKey] = []
34
35     for person, id in zip(["d", "m", "t"], [4,
36         13, 20]):
37         ldr.load(key_path(person))
            ldr.parse()

```

```

38             self.d_m_t_keys[id] = keys.RSAPrivateKey
39             ldr)
40             pubs.append(keys.RSAPublicKey(ldr))
41             self.d_pub, self.m_pub, self.t_pub = pubs
42             self.d_priv, self.m_priv, self.t_priv = self
43             .d_m_t_keys.values()
44             self.valid, self.missing_key, self.invalid
45             = LedgerLoader.load_from_csv(
46                 "./tests/test_transactions/mock_db.csv"
47             )
48             def test_add(self):
49                 """test adding transactions to a ledger"""
50
51                 with self.subTest("Add"):
52                     ledger = Ledger()
53                     self.assertFalse(not not ledger)
54                     ledger.append(Transaction(0, 0, 0, self.
55                     d_priv, self.m_priv))
56                     self.assertTrue(not not ledger)
57
58                 with self.subTest("Catch non transaction"),
59                 self.assertRaises(LedgerBuildError):
60                     ledger.append(6) # type: ignore
61
62             def test_load_from_csv(self):
63                 """Test loading from mock db works as
64                 expected"""
65                 ledger_list = LedgerLoader.load_from_csv(
66                     "./tests/test_transactions/mock_db.csv"
67                 )
68                 self.assertEqual(len(ledger_list), 3)
69
70             def test_verify_transactions(self):
71                 """
72                     Make three ledgers:
73                     1) Valid transactions
74                     2) An unsigned transaction
75                     3) An invalid signature

```

```
73
74         Check that first one goes through no issues
75         , and that other two are caught
76         """
77
78         with self.subTest("unsigned"), self.
79             assertRaises(VerificationError):
80                 self.valid._verify_transactions()
81
82
83         with self.subTest("signed"):
84             self.valid._verify_transactions()
85
86         with self.subTest("missing key"), self.
87             assertRaises(VerificationError):
88                 self.missing_key._verify_transactions()
89
90         with self.subTest("invalid key"), self.
91             assertRaises(VerificationError):
92                 self.invalid._verify_transactions()
93
94     def test_as_flow(self):
95         """Test that we build flow graphs from
96         ledgers as expected"""
97         # sign ledger
98         self.sign()
99
100        Vertex = src.simplify.graph_objects.Vertex
101
102        exp = src.simplify.flow_graph.FlowGraph(
103            [Vertex(ID=4), Vertex(ID=13), Vertex(ID
104            =20)])
105
106        d, m, t = exp.nodes()
107        exp.add_edge(d, (m, 10), (t, 5))
108        exp.add_edge(t, (m, 5))
109
110        as_flow = self.valid._as_flow()
```

```

107
108         with self.subTest("nodes"):
109             self.assertEqual(exp.nodes(), self.
110                           valid.nodes)
111
112         with self.subTest("to flow graph"):
113             self.assertEqual(exp, as_flow)
114
115     def sign(self):
116         """Checks that signing a ledger is a
117         reliable process"""
118         for trn in self.valid.ledger:
119             trn.sign(self.d_m_t_keys[trn.src],
120                      origin="src")
121             trn.sign(self.d_m_t_keys[trn.dest],
122                      origin="dest")
123             print("verifying...")
124             trn.verify()
125             print("verified")
126
127     def test_flow_to_transactions(self):
128         """Checks that flow graph data structures
129         can be effectively converted to ledgers"""
130         self.maxDiff = None
131         self.sign()
132         as_flow = self.valid._as_flow()
133
134         # remove sigs, ID, for comparison
135         trn: Transaction
136         for trn in self.valid.ledger:
137             trn.ID = 0
138             trn.signatures = {}
139
140         with self.subTest("to transactions"):
141             self.valid.ledger.sort(key=lambda trn:
142                                   trn.amount)
143             calc: list[Transaction] = self.valid.
144             _flow_to_transactions(as_flow)
145             calc.sort(key=lambda trn: trn.amount)
146
147             self.assertEqual(calc, self.valid.

```

```
140 ledger)
141
142     def test_simplify_ledger(self):
143         """Integration test, checks that calling
144             simplify() on a ledger displays expected behaviour
145         """
146
147         self.sign()
148         self.valid.simplify_ledger()
149
150         trn = Transaction(4, 13, 15, self.d_pub,
151                           self.m_pub)
152
153         self.assertEqual(self.valid.ledger, [trn])
```



```
1 # coding=utf-8
2 import os
3 import unittest
4
5 from src.transactions.transaction import *
6
7
8 class TestTransaction(unittest.TestCase):
9     def setUp(self) -> None:
10         # Load keys
11         os.chdir("/home/tcassar/projects/settle/src")
12
13         ldr = keys.RSAKeyLoader()
14         ldr.load("./crypto/sample_keys/d_private-key.pem")
15         ldr.parse()
16
17         self.key = keys.RSAPrivateKey(ldr)
18         self.pub_key = keys.RSAPublicKey(ldr)
19
20         self.trn = Transaction(0, 0, 0, self.pub_key,
21                               self.pub_key)
22         self.trn.time = 0 # type: ignore
23
24     def test_hash(self):
25         """Tests that calling .hash() on a
26         Transaction displays expected behaviour"""
27         self.assertEqual(
28             self.trn.hash(),
29             b"\xb9\x8f\xe6\xde\x08\xd4\x1f\xdd\x01\x
30             \xa3\xb4\xc5\x1d\x98\xd4\xac\x1b\x02\xd3\x
31             \xa0\x01!\xf3\x99\xdf\xd5\\x85",
32             )
33         self.trn.time = 1
34         self.assertNotEqual(
35             self.trn.hash(),
36             b"\xb9\x8f\xe6\xde\x08\xd4\x1f\xdd\x01\x
37             \xa3\xb4\xc5\x1d\x98\xd4\xac\x1b\x02\xd3\x
38             \xa0\x01!\xf3\x99\xdf\xd5\\x85",
39             )
```

```

34     def test_sign(self):
35         """
36             Checks that signing a Transaction displays
37             expected behaviour
38             Working on assumption that rsa.Notary is
39             working; tested in settle/tests/test_crypto"""
40
41         self.trn.sign(self.key, origin="src")
42
43         with self.subTest("catch invalid origin"),
44             self.assertRaises(ValueError):
45                 self.trn.sign(self.key, origin="no")
46
47         with self.subTest("invalid key types"), self.
48             .assertRaises(TransactionError):
49                 self.trn.sign(12, origin="src")
50                 self.trn.sign(self.pub_key, origin="dest")
51
52         with self.subTest("sig_overwrite"), self.
53             .assertRaises(TransactionError):
54                 self.trn.sign(self.key, origin="src")
55
56         with self.subTest("Right sig"):
57             self.assertEqual(
58                 b"\xc1~\xcb\u\xec\x01E*\xf2;0\xe3\
59                 \xf3\x08x\xee\x84\xfc\xe1\xca\x8b\xa2\xed\xec\x9b\
60                 \xfd\xe5$7\x88n\xb7\x86\xfc~\x98\x91\x80Z\xcd\x1e\xf5\
61                 }\xc2<JS\xefY\xe5UW\xfc\x0e\xd2\xbe\xc9\xea\xfbZm\
62                 \xf2\xa7\x08\x8d\x05\x16\xe4@\xf40\x03I\xca\xbc.\x95\
63                 \xbb\xd3\n\xfd9\xb0Wk\xf4\x03\x96\xdbF\xcd\xa0E\xd1\
64                 \xac\xa6(\xb9\x92\xdb\x841\xe0U"\xe4\x7f\xeb U\xc9Z\
65                 \xe5\xf6\x19\xc1Wn\x17&\x17\x84Dt\xb6Z\xc0\x02\x85\
66                 \xf3\xd3\xd5\x98t7\xcd_\xfc\xa7\\xa7t\xe8\xb1\xafL\
67                 \x05\xb3\x07a\xd0jr\xc4}\xd8\xb58\xf40\x9f\x02\xa2\
68                 \xe6?3B\xf6\xe1\xe6\xb2\x02d\xfd\xd5\x83\xcf\xcc\xb6\
69                 \x06m\xc2p\xd7\x00@\\x93H\xc6]\x0c\x98\x1e\xdf\xda\x00\
70                 \x86\xd7\xec\xc7\x10\x025eiry\xd6\x80\xfe\xe2+\xb8\
71                 \x1dn\noB\xc0\x8a\xc8t\xbfg\xbb\xca(Aj\x85\xeb\x94\
72                 \x0c-\xacr'\xfe\n^Z\x13\x80'\xaa\x96{\xf8x\xce\xd6\
73                 \x90",

```

```
54                     self.trn.signatures[self.trn.src],  
55                 )  
56  
57     def test_verify(self):  
58         """Checks that we are able to verify  
transactions"""  
59         # check that we are complained at if no  
valid keys are passed in  
60         with self.subTest("catch invalid keys"),  
self.assertRaises(VerificationError):  
61             self.trn.verify()  
62             self.trn.verify() # wrong type # type  
: ignore  
63  
64         # check works with priv and public keys  
65  
66         with self.subTest("good verif"):  
67             self.trn.sign(self.key, origin="src")  
68             self.trn.verify()  
69  
70         with self.subTest("priv/pub keys"):  
71             self.trn.verify()  
72             self.trn.verify()  
73  
74         with self.subTest("verify src, dest"):  
75             self.trn.verify()  
76             self.trn.verify()  
77  
78         with self.subTest("verify >1 param"):  
79             self.trn.verify()  
80  
81         with self.subTest("bad key"), self.  
assertRaises(VerificationError):  
82             # edit pub key, thus should fail  
83             self.pub_key.lookup["n"] = 3  
84             self.trn.verify()  
85
```