Settle

A-Level Computer Science NEA

Thomas Cassar
Candidate Number 4037

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**.

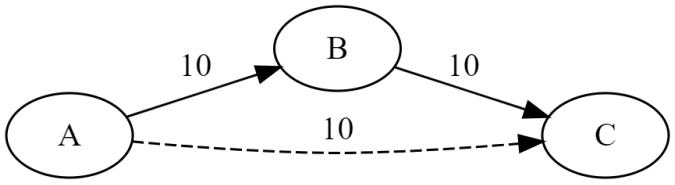
The End User

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. It's 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.

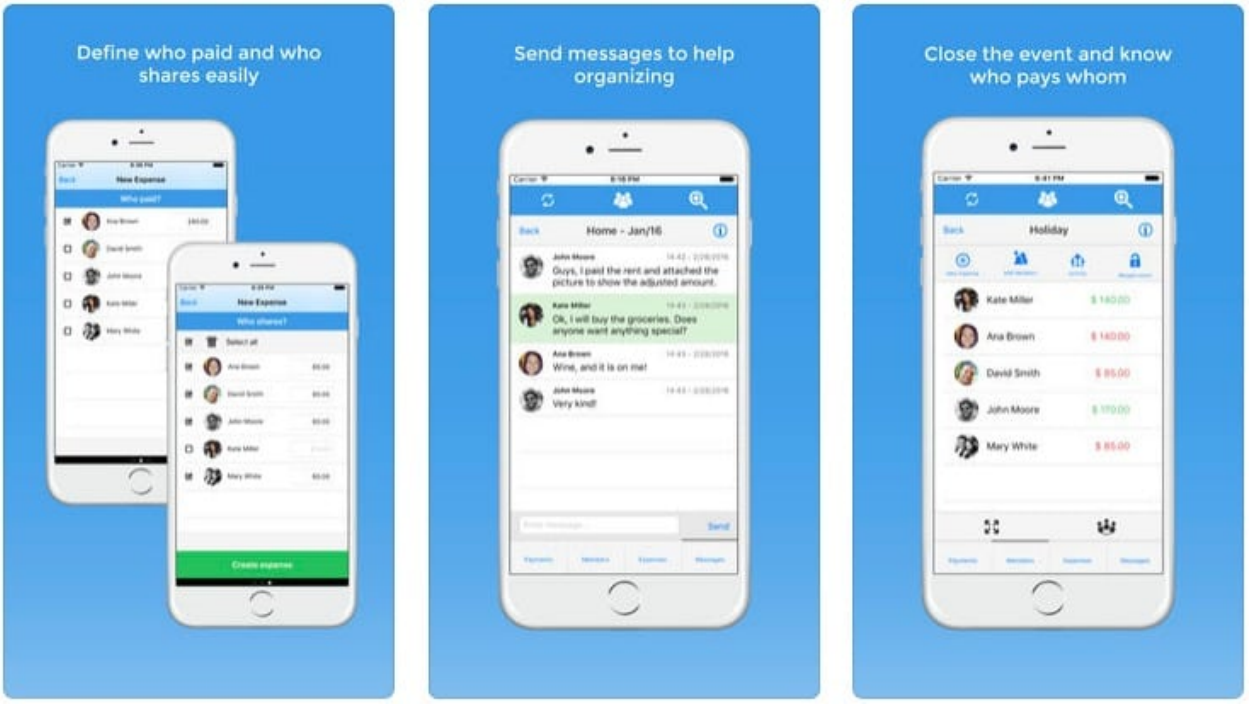
This is all accurate at time of writing.

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.

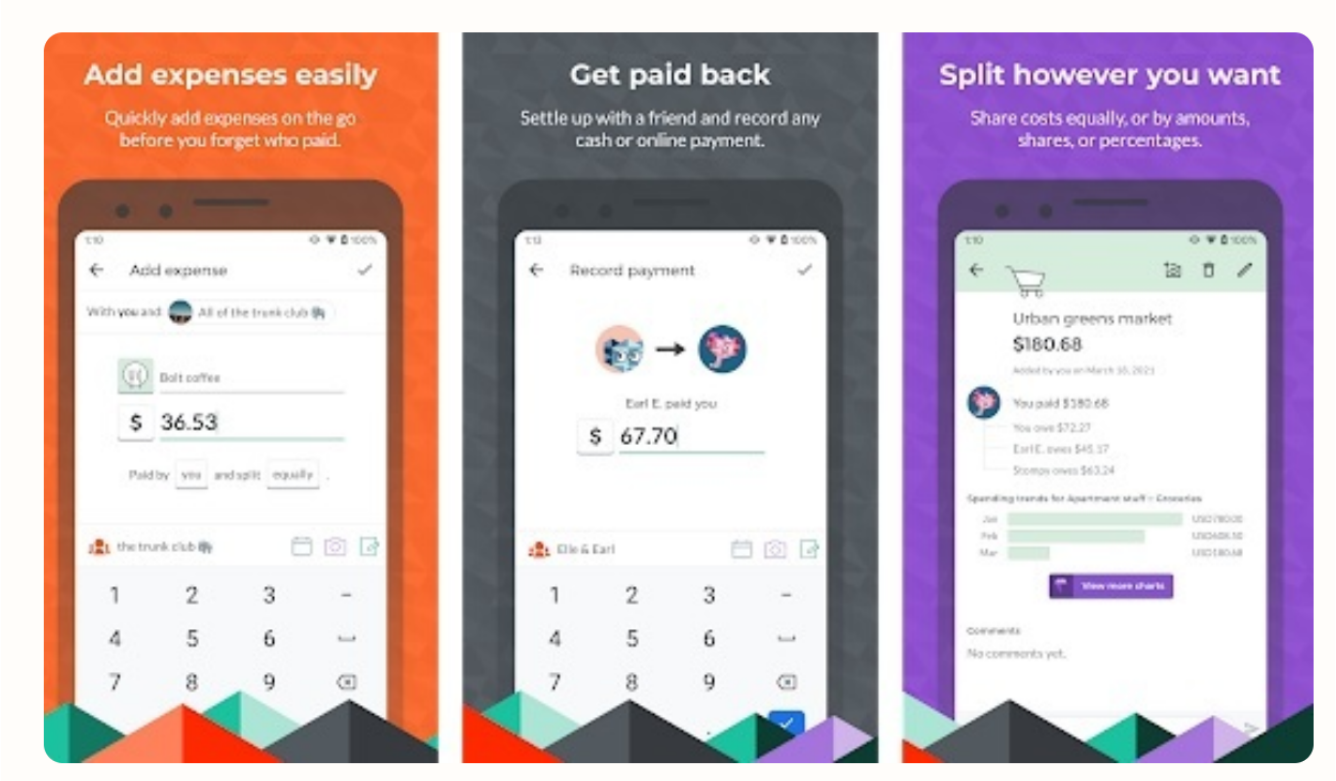


Evenfy also allows for a group to settle debts easily. This feature will be discussed more in the next product evaluation, as an identical feature appears there.

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 $\mathcal{O}(logV)$ 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
                        giver 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-
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.

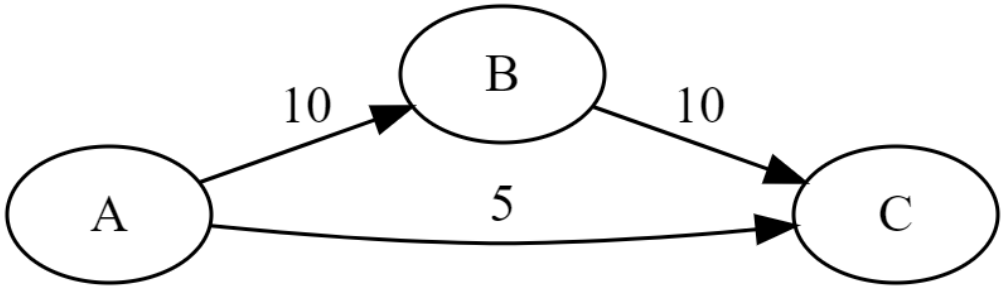
Another problem with this approach is that it contradicts one of my initial high level requirements. It does not preserve any sense of who owes whom past step 1. Thus, it cannot guarantee that no one will owe anyone that they did not previously owe.

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 -[10]→ B -[10]→ 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.

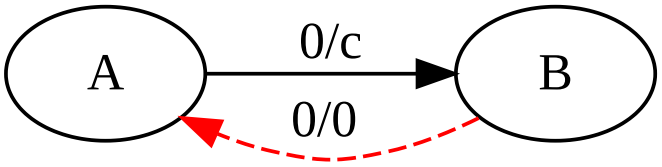
The way in which the algorithm works is simple:
1) Find an augmenting path from source node to sink node (through the residual graph)
2) Augment flow down path
3) Repeat until no more augmenting paths exist

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).

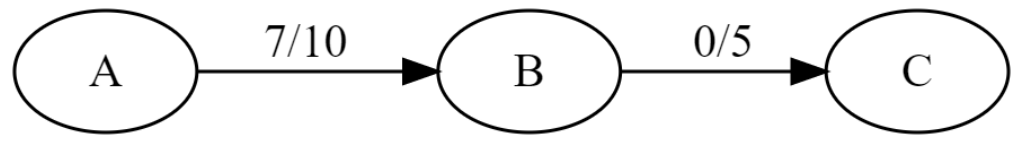
Augmenting Path
The augmenting path is a path of edges in the residual graph, where each edge has a remaining capacity > 0. The path is from two specified nodes - a source node s and a sink node t.

Residual Graph and **Residual Edges**
The residual graph is the combination of the flow graph , and residual edges. For each original edge from u → v , with capacity c, there exists a residual edge from v → u, with capacity 0.

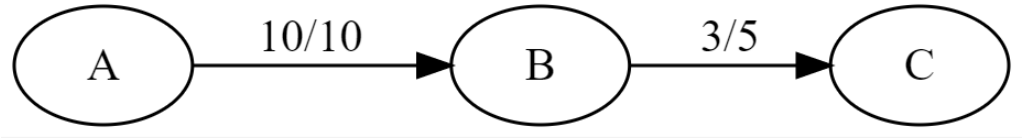


Residual edges are valid edges to consider when looking for an augmenting path, given that they have unused capacity (the above example's residual edge has an unused capacity 0, as $0 - 0 = 0$).

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 →B has 3 units of unused capacity, and B → 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 → B would have a capacity of -3.

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

poOnce 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]].

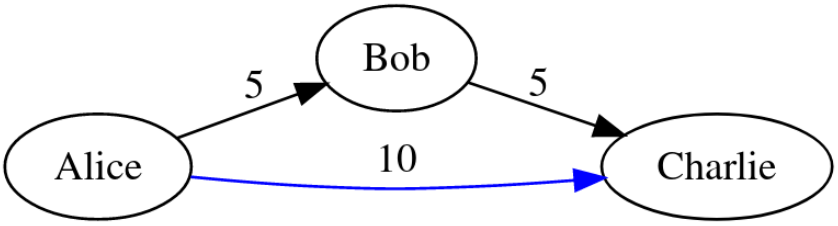
Since Edmonds-Karp's runtime is independent of flow, its input, it is classed as a strongly polynomial algorithm, making it perform much better than the Fulkerson-Ford max flow algorithm. Thus, this is the algorithm that I will be implementing to simplify debts across a group.

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.

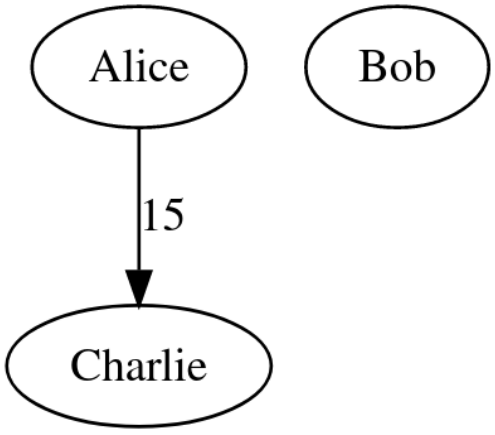
After researching, I found a solution which proposed the following.

```
// 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)
```



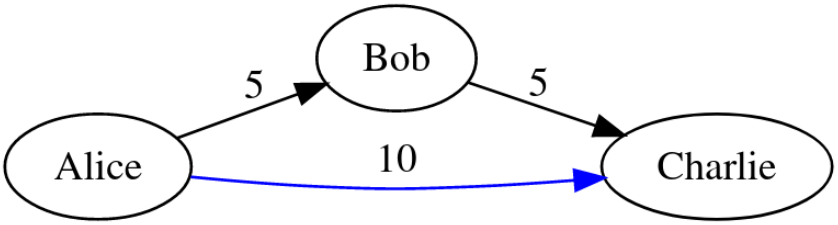
In prose, a new weighted digraph is generated (with no edges, but all the same nodes) for the cleaned edges.

Starting on node G, we would therefore run a max flow from $G \to B$. If the max flow from $G \to B > 0$, then an edge with the weight of the max flow from $G \to B$ is added to the new weighted digraph. The edge from $G \to B$ in the flow graph is deleted. This process will happen again from $G \to D$. 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]



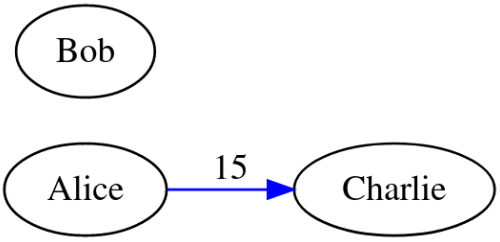
However, while hand tracing the aforementioned algorithm, I discovered that it was not a correct algorithm.

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

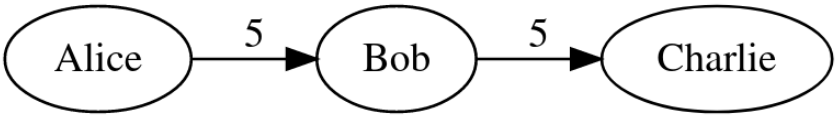


Running Edmonds-Karp along this network with Alice as a source and Charlie as a sink gives a max-flow of 15. Thus, add an edge to the new graph from Alice to Charlie with a weight of 15, and remove the Alice - Charlie edge from the initial graph

This gives the new graph:

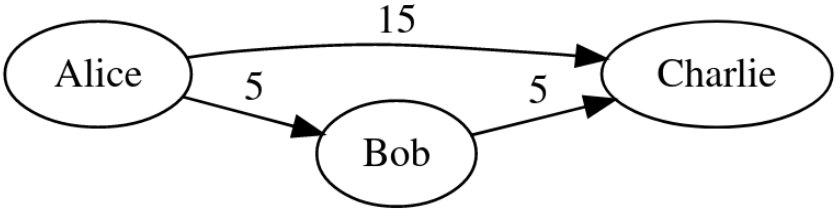


And the 'initial' graph



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.

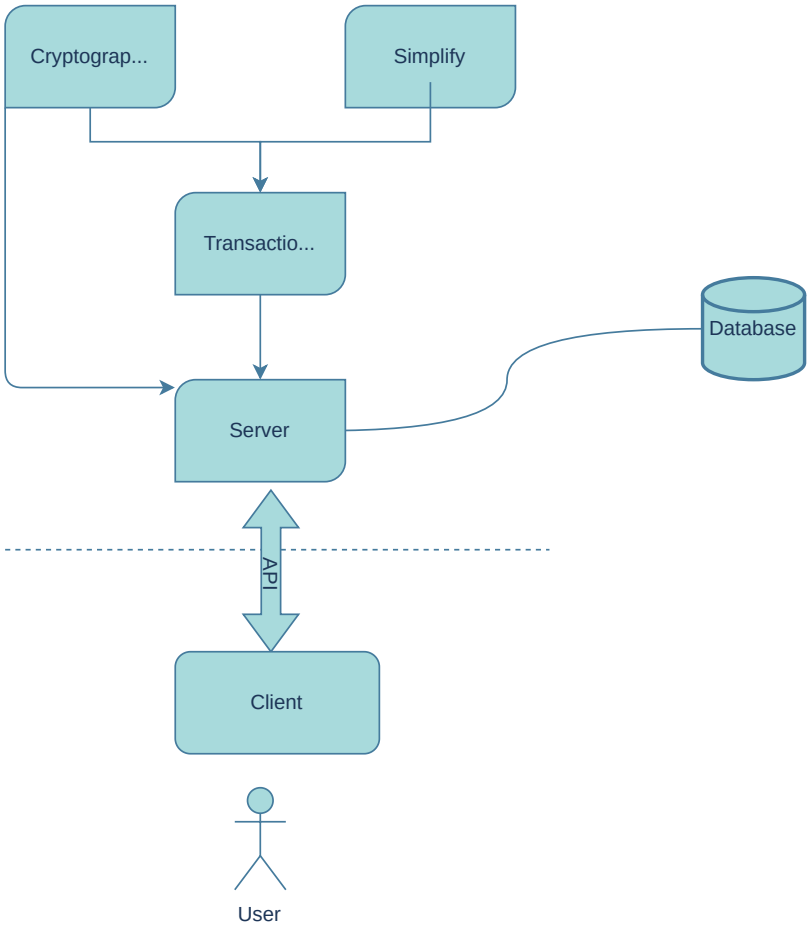
Next comes the transactions themselves. Only at the transaction level should the concepts of cryptography or the graph processing start being used.

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
```

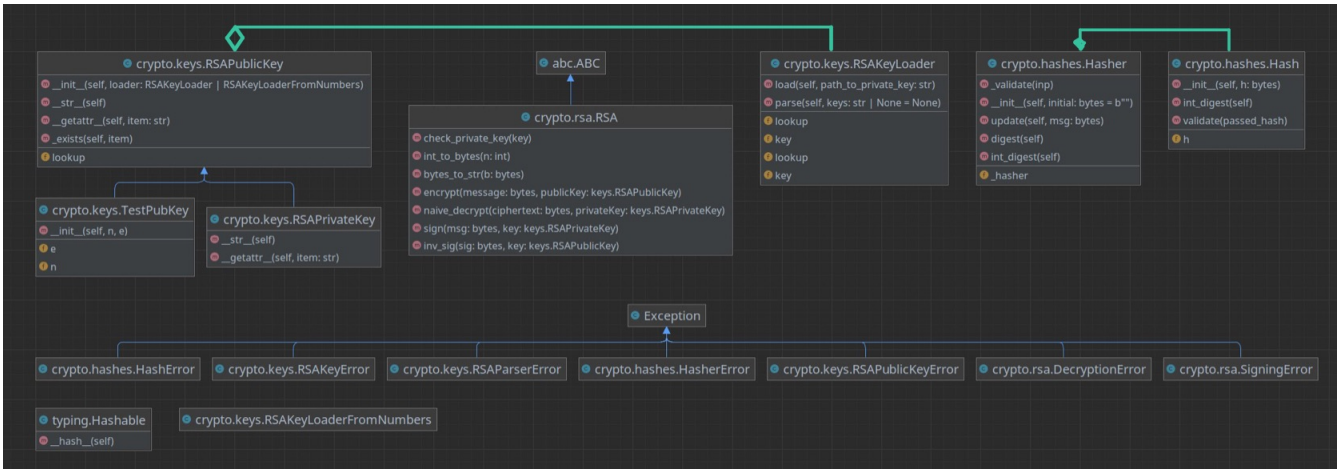
```
├── setup.py
├── src
│   ├── client
│   │   ├── client.py
│   │   ├── cli_helpers.py
│   │   ├── cli.py
│   │   └── __init__.py
│   ├── crypto
│   │   ├── hashes.py
│   │   ├── __init__.py
│   │   ├── keys.py
│   │   ├── rsa.py
│   │   └── sample_keys
│   │       ├── d_private-key.pem
│   │       ├── 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.

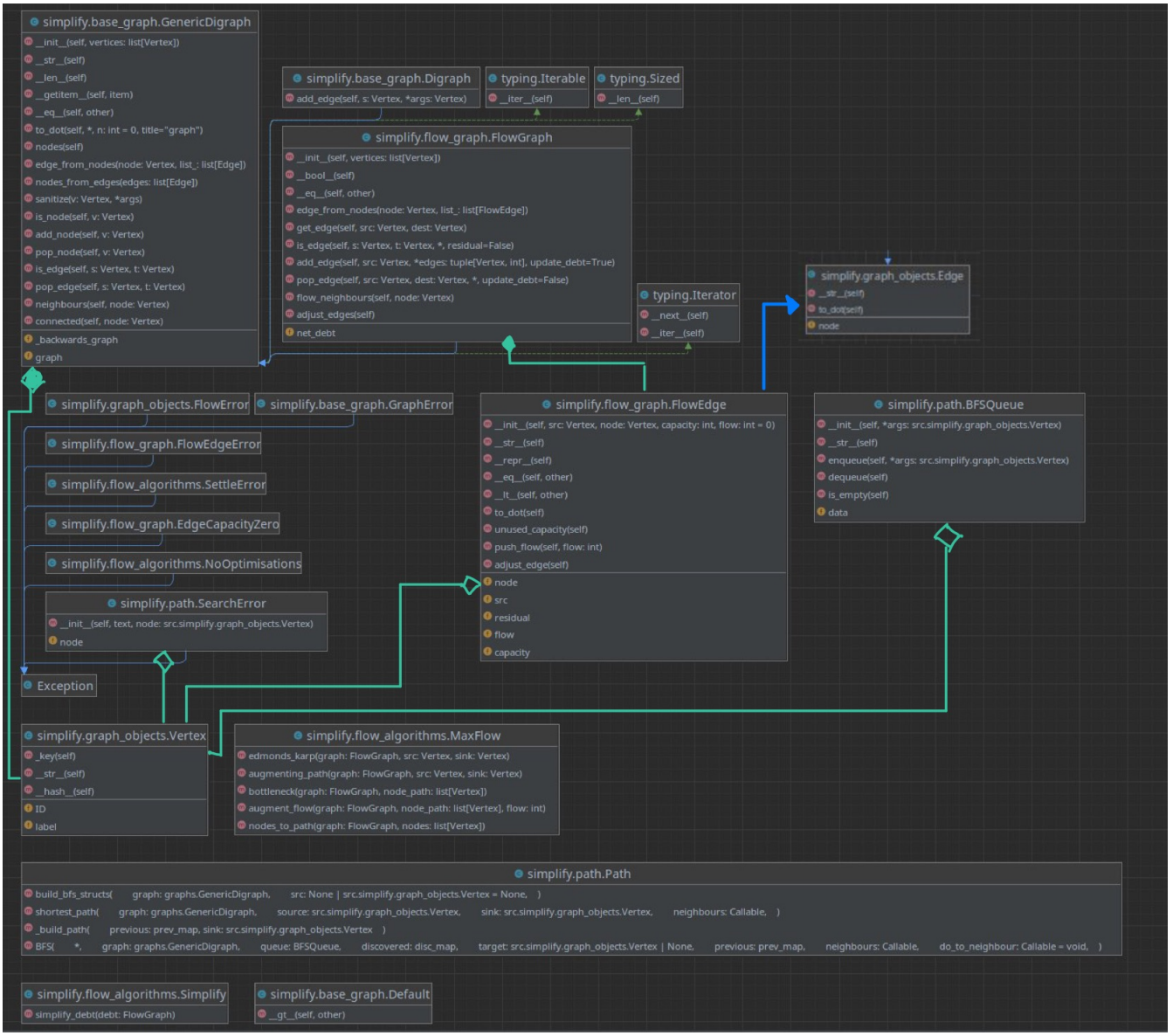
Thus, it is possible to ensure validity, and verify the origin, of messages.

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 → B of weight 5, and an edge from B → 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.

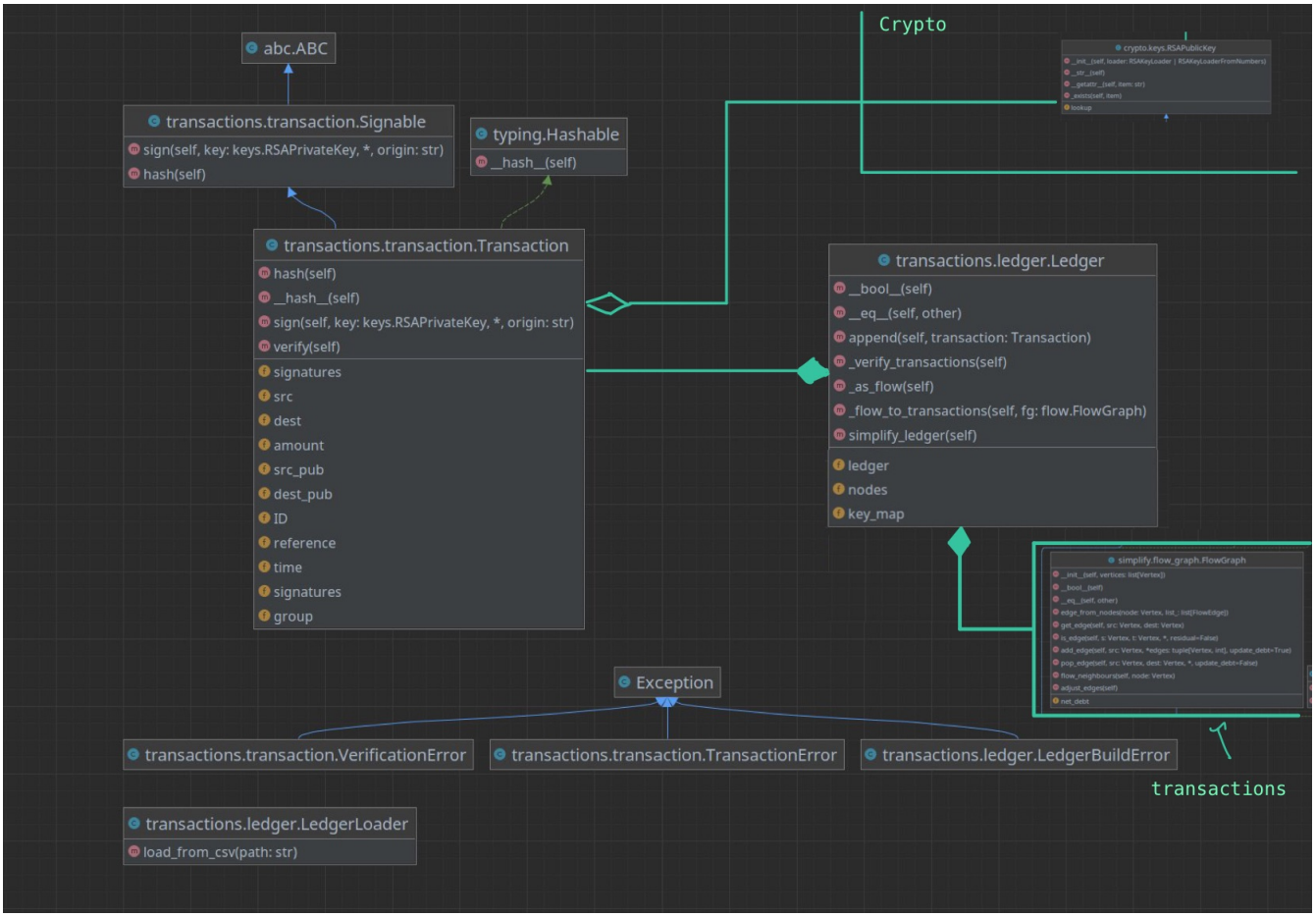
In pseudocode

```
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.

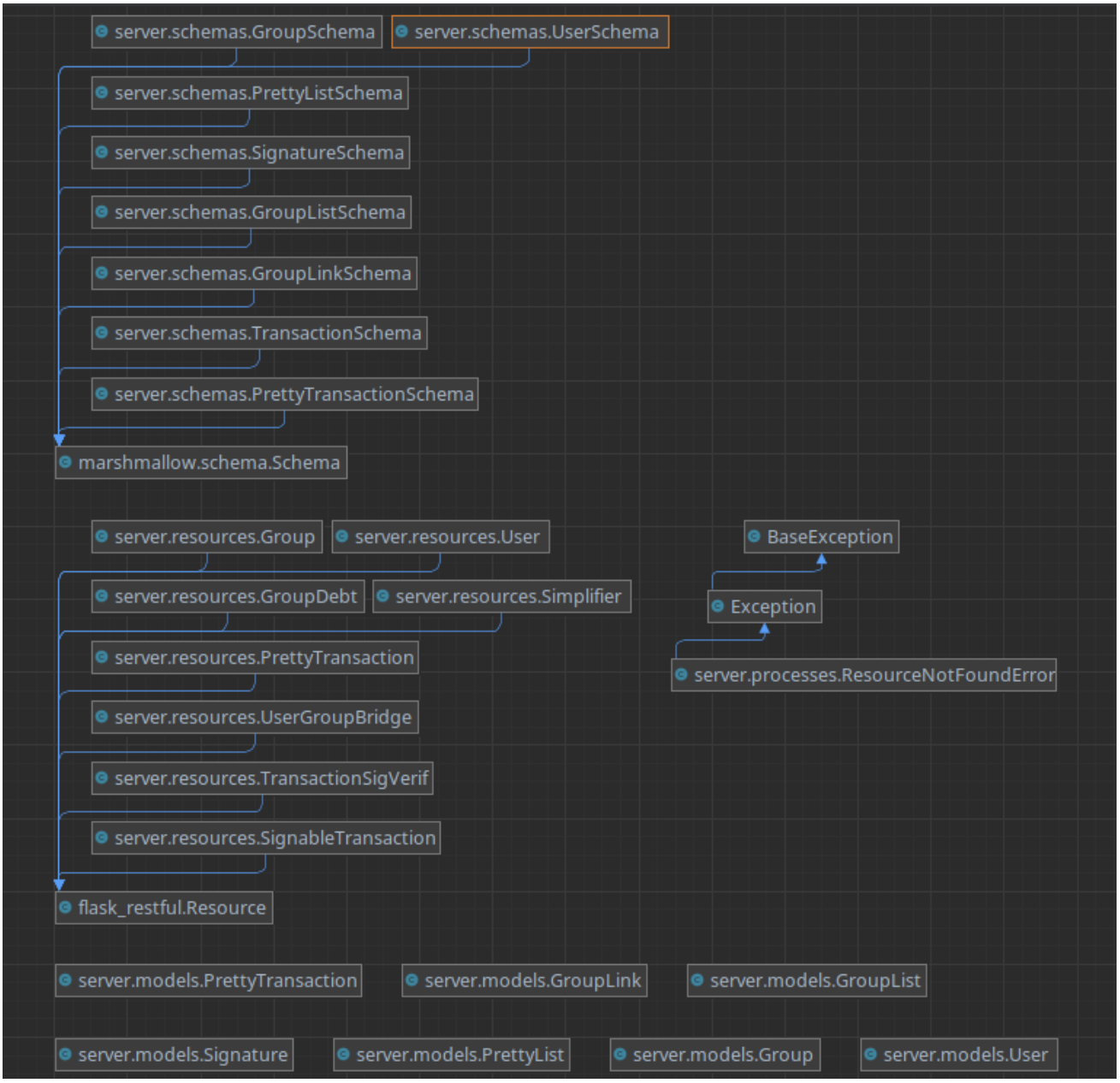
Signatures are stored as bytes inside the transaction object. Time is stored as a `datetime.datetime` object, and keys are stores as `crypto.keys.RSAPublicKey` objects (as per class diagram).

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.

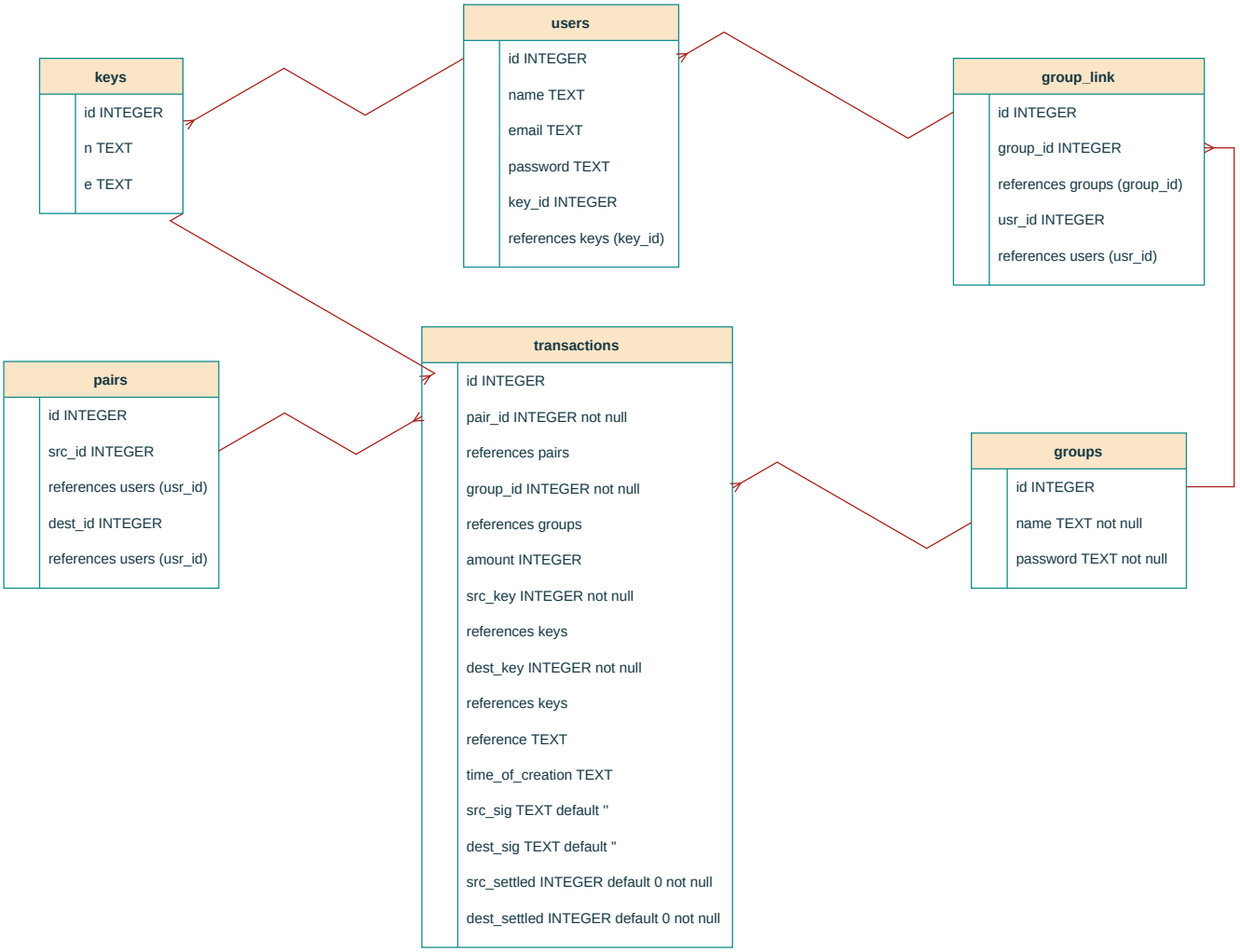
`PrettyTransaction` is a transaction object that is intended for being viewed on the front end by a user. It has people saved as emails as opposed to IDs, an association with a group, no keys involved, the time of creation, reference, and verification status. It also has the transaction ID. The POST 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

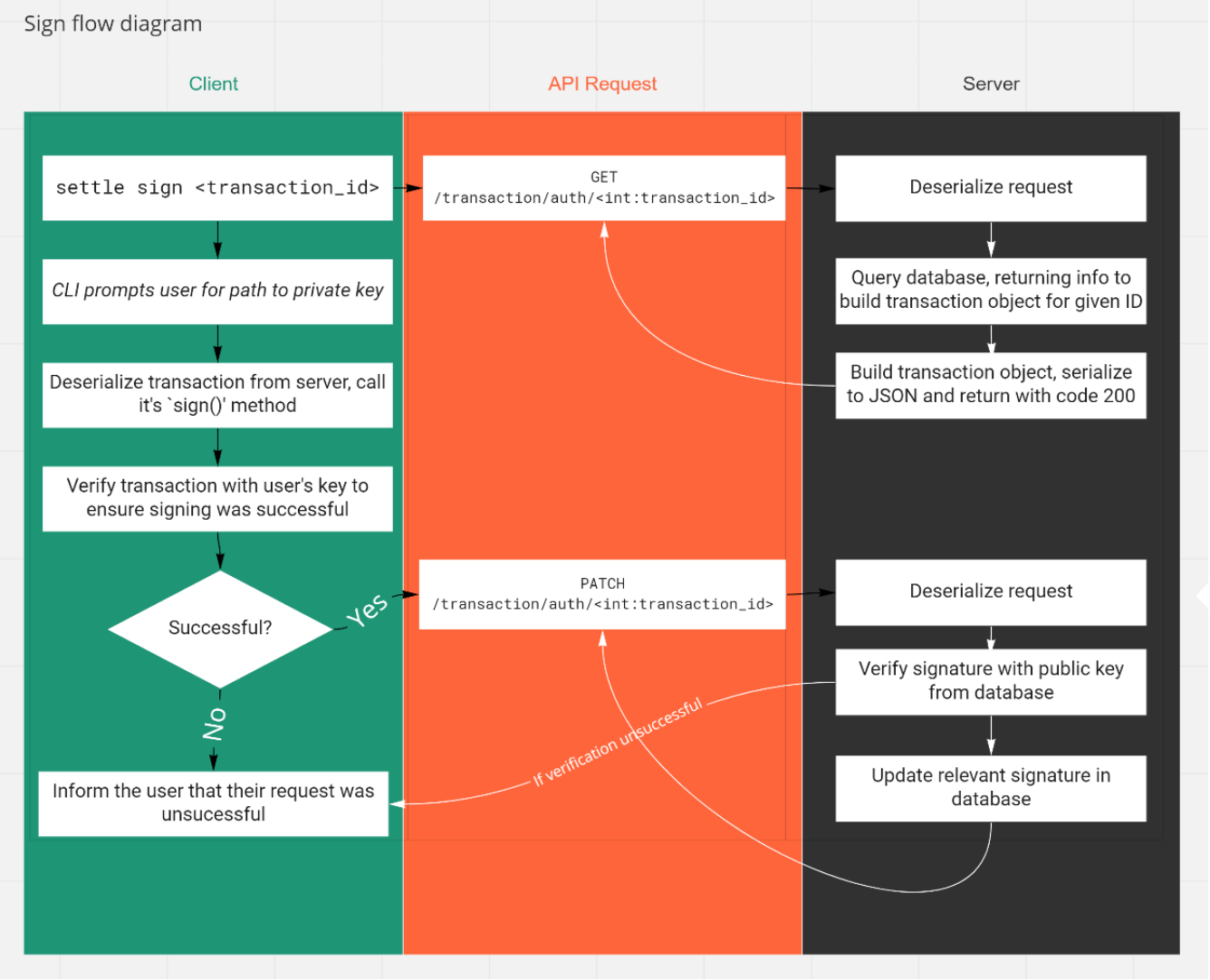
Server Logic

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.

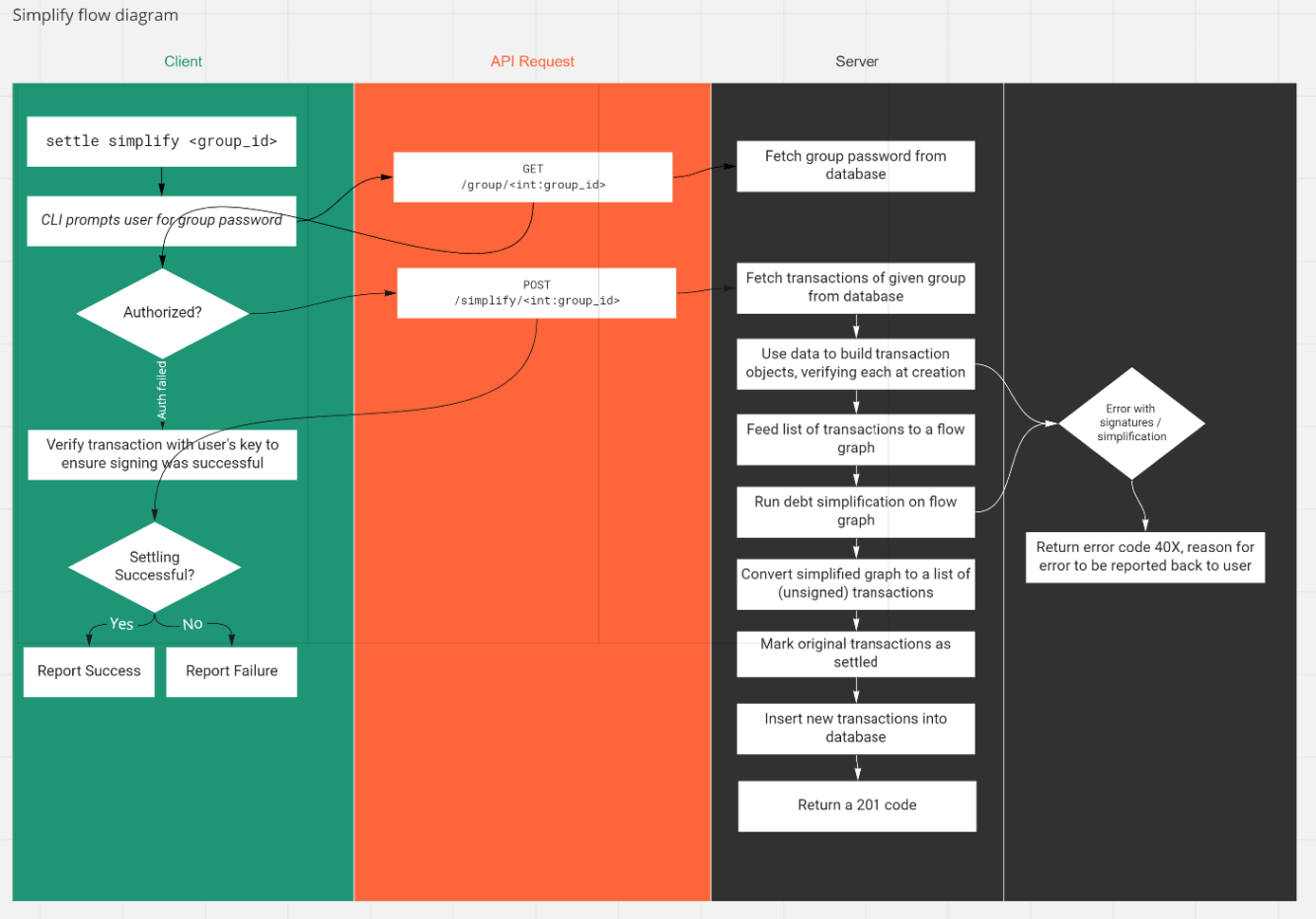
All client-server communication is done with JavaScript Object Notation (JSON) for this project.

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 = ?;
```

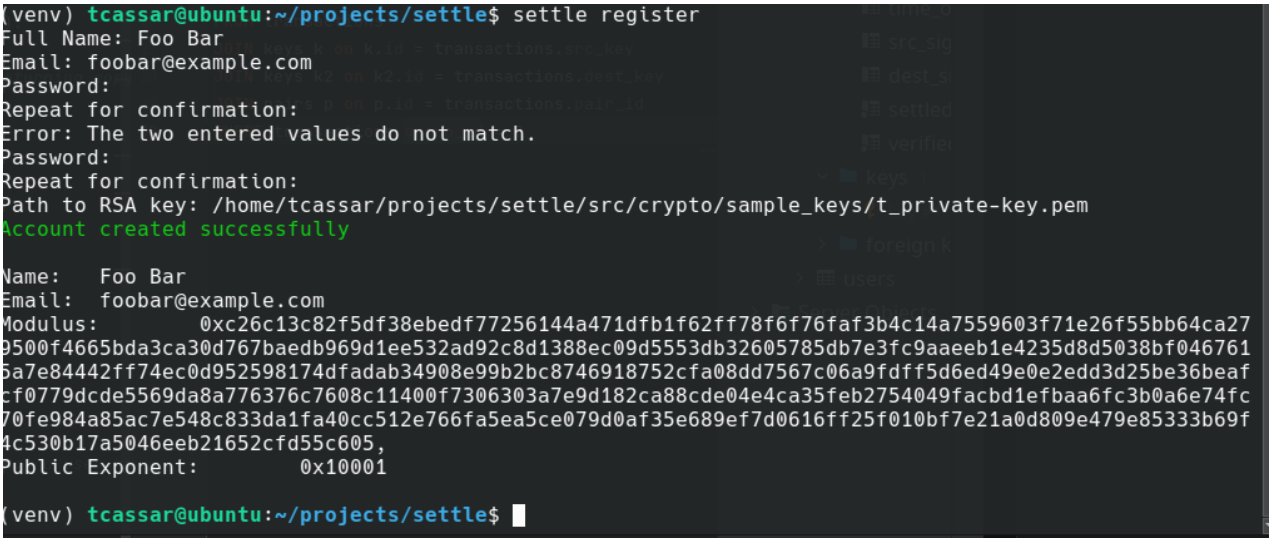
to get all the data needed to build a `transaction.ledger.Ledger` of `transaction.transaction.Transaction` objects. Once those objects are build, I call `ledger.simplify_ledger()`. This will then invoke all the logic discussed in the `simplify` module section, as well as handle the verification of signatures, 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.

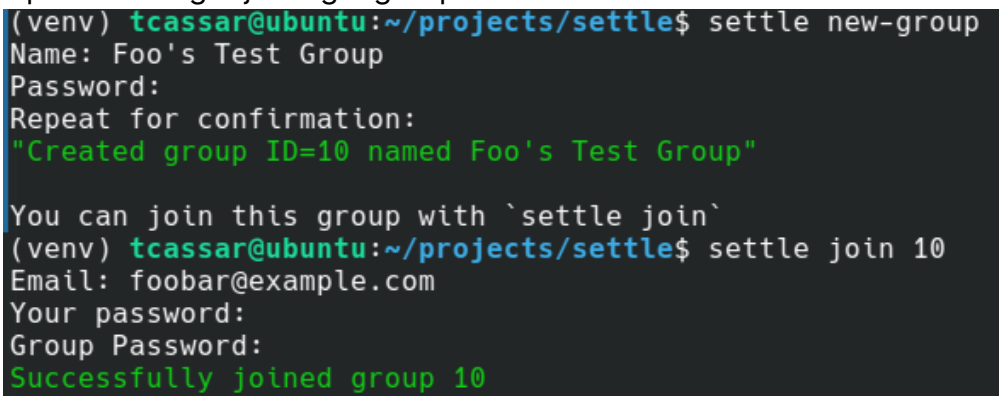
To show this, I will provide screenshots of an output to STDOUT of how I would like certain outputs to look. Here I will provide mainly ancillary outputs. and how I would like certain prompts to appear upon a command being run

- Upon a user registering a new account

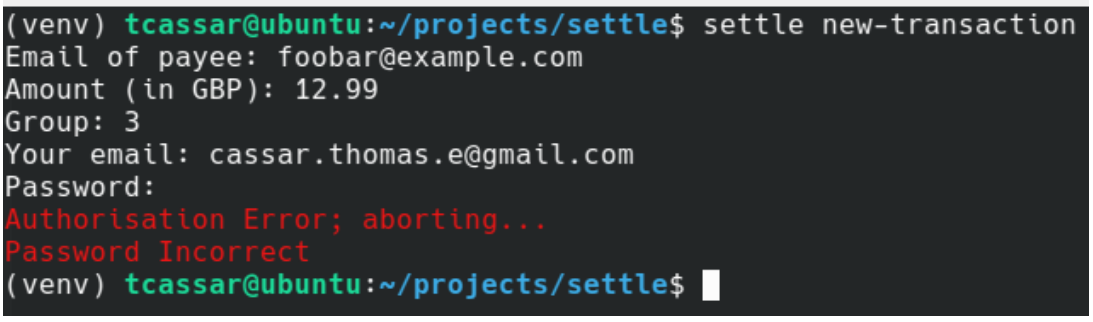


The program should prompt the user with the details that they need to enter to create their account. Password entering should be hidden, as it is commonly in CLIs.
Passwords should be confirmed through asking for confirmation, as above. If the passwords entered do not match, the program should ask the 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



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



- 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
```

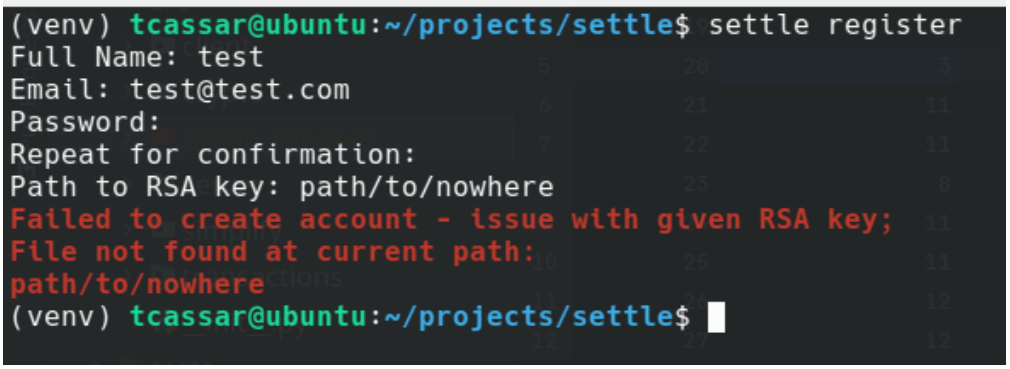
This text should also be displayed when the `--help` flag is called after any of the commands.

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.



---

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

```python
# coding=utf-8
import logging

logging.basicConfig(filename='./log')
```

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

```python
# coding=utf-8

import click

import src.client.client as client


@click.group()
def settle():
    ...


# |--------|
# |  USER  |
# |--------|


@click.option("--pub_key", prompt="Path to RSA key", type=click.Path())
@click.option("--password", prompt=True, hide_input=True, confirmation_prompt=True)
@click.option("--email", prompt=True)
@click.option("--name", prompt="Full Name")
@settle.command()
def register(
    name,
    email,
    password,
    pub_key,
):
    client.register(name, email, password, pub_key)


@click.argument("email")
@settle.command()
def whois(email):
    client.whois(email)


@click.option("-g", "--groups", flag_value="groups", 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,
    group_password)
 88
 89
 90 @click.option("--password", prompt="Group Password"
    , 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
    =True)
 99 @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
    payee")
104 @settle.command(name="new-transaction")
105 def new_transaction(email, password, dest_email,
    amount, group, reference):
106     """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)
```

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

```python
# coding=utf-8
import copy
import sys

import click
import json
import requests

import src.client.cli_helpers as helpers
import src.crypto.keys as keys
import src.server.models as models
import src.server.schemas as schemas
import src.transactions.transaction as trn
from src.client.cli_helpers import show_transactions

trap = helpers.trap

###############
#  ANCILLARY  #
###############


@trap
def register(
    name,
    email,
    password,
    pub_key,
):
    """Register to settle, using email, passwd, and an RSA public key"""

    # extract and store modulus and pub exp as bytes, ready for db
    ldr = keys.RSAKeyLoader()
    try:
        ldr.load(pub_key)
        ldr.parse()
        pub_key: keys.RSAPublicKey = keys.RSAPublicKey(ldr)
    except keys.RSAParserError as rsa_err:
```

```
39          click.secho(
40              f"Failed to create account - issue with
   given RSA key;\n{rsa_err}",
41              fg="red",
42              bold=True,
43          )
44          return
45
46      password: str = helpers.hash_password(password)
47
48      usr = models.User(
49          name,
50          email,
51          hex(pub_key.n),
52          hex(pub_key.e),
53          password,
54      )
55
56      # build json repr of object
57      schema = schemas.UserSchema()
58      usr_as_json = schema.dump(usr)
59
60      response = requests.post(helpers.url("user"),
   json=usr_as_json)
61      try:
62          helpers.validate_response(response)
63      except helpers.InvalidResponseError:
64          print(response)
65          click.secho(f"Failed to create account under
    email {email}", fg="yellow")
66          return
67
68      click.secho(f"Account created successfully", fg=
   "green")
69      click.echo(f"{usr}")
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
     along with IDs"""
 89
 90     # note: flags are None or True for some
    godforsaken reason
 91
 92     # show both if no flags
 93     if not transactions and not groups:
 94         transactions = groups = True
 95
 96     if groups:
 97         click.secho("Groups that you are a member
    of:\n", fg="blue")
 98
 99         # receive list of groups JSON;
100         try:
101             groups_data = requests.get(helpers.url(
    f"group/{email}"))
102         except helpers.ResourceNotFoundError as ire
    :
103             raise helpers.ResourceNotFoundError(
104                 f"Problem fetching your group...\n{
    ire}"
105             )
106
107         group_objs: list[models.Group] = []
108         for group in groups_data.json()["groups"]:
```

```
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 {
    group_id}", fg="green")
146 
147 
148 @trap
149 def new_group(name, password):
150     schema = schemas.GroupSchema()
151     # note: 0 is placeholder, will be overwritten
    by db
152     group = models.Group(0, name, helpers.
    hash_password(password))
153 
154     as_json = schema.dump(group)
155     response = requests.post(helpers.url("group"),
    json=as_json)
156 
157     try:
158         helpers.validate_response(response)
159     except helpers.InvalidResponseError as ire:
160         raise helpers.InvalidResponseError(f"Couldn
    't create new group...\n{ire}")
161 
162     click.secho(response.text, fg="green")
163     click.secho("You can join this group with `
    settle join`")
164 
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
    destination exists
181     src = helpers.get_user(email)
182     dest = helpers.get_user(dest_email)
183 
184     # convert amount into pence
185     amount = float(amount)
186     amount *= 100
187     if type(amount) == float:
188         amount //= 1
189         amount = int(amount)
190 
191     # build key objects
192     src_key_ldr = keys.RSAKeyLoaderFromNumbers()
193     dest_key_ldr = keys.RSAKeyLoaderFromNumbers()
194 
195     src_key_ldr.load(n=int(src.modulus, 16), e=int(
    src.pub_exp, 16))
196     dest_key_ldr.load(n=int(dest.modulus, 16), e=
    int(dest.pub_exp, 16))
197 
198     src_key = keys.RSAPublicKey(src_key_ldr)
199     dest_key = keys.RSAPublicKey(dest_key_ldr)
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()
```

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

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

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

```
313 the users in the transaction"
314         )
315 
316     try:
317         # convert keys of sigs to ints to enable
    overwrite protection
318         as_strs = copy.deepcopy(transaction.
    signatures)
319         transaction.signatures = {}
320         for s_key, s_val in as_strs.items():
321             if type(s_key) is str:
322                 transaction.signatures[int(s_key
    )] = s_val
323 
324         transaction.sign(key, origin=origin)
325         click.secho("\tsuccessfully signed
    transaction", fg="green")
326     except trn.TransactionError as te:
327         raise helpers.AuthError(f"Failed to sign
    transaction: {transaction.ID}\n{te}")
328 
329     # convert signature to hex
330     # todo: add sig_as_hex to transaction obj
331     sig_hex = hex(int.from_bytes(transaction.
    signatures[usr.id], sys.byteorder))
332 
333     # patch signature
334     schema = schemas.SignatureSchema()
335     signature = models.Signature(transaction_id,
    sig_hex, origin)
336     rep = requests.patch(helpers.url("transaction/
    auth/"), json=schema.dump(signature))
337 
338     if rep.status_code == 201:
339         click.secho("\tSuccessfully appended
    signature in database!", fg="green")
340 
341 
342 @trap
343 def verify(groups, transactions: int):
344     """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
    transaction ID", fg="yellow")
348         return
349
350     if transactions:
351         response = requests.get(helpers.url(f"
    transaction/auth/{transactions}"))
352
353         try:
354             helpers.validate_response(response)
355         except helpers.InvalidResponseError as ire:
356             raise helpers.InvalidResponseError(f"
    Error in getting transaction, {ire}")
357
358         schema = schemas.PrettyTransactionSchema()
359         schema.make_pretty_transaction(response.
    json()).secho()
360
361     if groups:
362         try:
363             transactions_data = requests.get(
    helpers.url(f"group/debt/{groups}"))
364             helpers.validate_response(
    transactions_data)
365         except helpers.ResourceNotFoundError as ire
    :
366             raise helpers.ResourceNotFoundError(
367                 f"Problem with fetching your
    transactions...\n{ire}"
368             )
369
370         pretty_schema = schemas.PrettyListSchema()
371         # print(transactions_data.json())
372         pretty_list: models.PrettyList =
    pretty_schema.make_pretty_list(
373             transactions_data.json()
374         )
375         trn_schema = schemas.
```

```
375 PrettyTransactionSchema()
376
377         for trn in pretty_list.src_list +
    pretty_list.dest_list:
378             trn_schema.make_pretty_transaction(trn
    ).secho()
379
380
381 @trap
382 def tick(email: str, password: str, t_id: int):
383     # auth user
384     helpers.auth_usr(email, password)
385
386     rep = requests.patch(helpers.url(f"transaction/
    settle/{t_id}"), json=json.dumps({'email': email},
    indent=4))
387     helpers.validate_response(rep)
388     click.secho(f'Transaction {t_id} marked as
    settled!', fg='green')
389
390
```

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

```python
# coding=utf-8
import logging

logging.basicConfig(level=logging.CRITICAL)

```

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

```python
# coding=utf-8
import click
import requests

from src.crypto import hashes as hasher
from src.server import models as models, schemas as schemas

SERVER = "http://127.0.0.1:5000/"


class AuthError(Exception):
    ...


class ResourceNotFoundError(Exception):
    """Resource could not be found on the server (404)"""


class InvalidResponseError(Exception):
    """Requested action was understood but not allowed (403 / 409)"""


class ServerError(Exception):
    ...


class NoChanges(Exception):
    ...


def hash_password(password) -> str:
    return str(hasher.Hasher(password.encode(encoding="utf8")).digest().h)


def url(query: str) -> str:
    return SERVER + query

```

```
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     return _auth(f"user/{email}", password)
 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"],
    usr["pub_exp"], "", usr["id"]
 96     )
 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;
    aborting...", fg="red")
110             click.secho(ae, fg="red")
111
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
    }", fg="yellow")
123
124         except requests.exceptions.ConnectionError:
125             click.secho('ERROR: Could not connect
    to server', fg='red')
126
127     return inner
128
129
130 def show_transactions(transactions_data: requests.
    Response):
131     try:
132         unverified_running = 0.00
133         verified_running = 0.00
134         for pretty in transactions_data.json()["
    src_list"]:
135
136             click.secho(
137                 f'\nYou owe {pretty["other"]} £{
    round(pretty["amount"] / 100, 2):.2f}',
138                 fg="yellow",
139             )
140
141             click.secho(
142                 f'\nReference: {pretty["time"]}'
143                 + 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["
    amount"]
153 
154         for pretty in transactions_data.json()["
    dest_list"]:
155             click.secho(
156                 f'\n{pretty["other"]} owes you £{
    round(pretty["amount"] / 100, 2):.2f}',
157                 fg="yellow",
158             )
159 
160             click.secho(
161                 f'\nReference: {pretty["time"]}'
162                 + f'\nAgreed upon at {pretty["
    reference"]}'
163                 + f'ID: {pretty["id"]}'
164             )
165 
166             if pretty["verified"] == 1:
167                 click.secho("Verified", fg="green")
168                 verified_running -= pretty["amount"
    ]
169             else:
170                 click.secho("Unverified", fg="red")
171                 unverified_running -= pretty["
    amount"]
172 
173         click.echo("----------\n")
174 
175         unverified_running = round(
    unverified_running / 100, 2)
```

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

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

```python
# coding=utf-8
"""
RSA Sign/Verify classes
"""

from __future__ import annotations

import sys
from abc import ABC
from typing import TYPE_CHECKING

from src.crypto import keys

if TYPE_CHECKING:
    pass


class DecryptionError(Exception):
    """Failed to decrypt"""


class SigningError(Exception):
    ...


class RSA(ABC):
    """RSA methods; ABC so is never instantiated; just used for namespacing"""

    @staticmethod
    def check_private_key(key) -> None:
        if type(key) == keys.RSAPublicKey:
            raise DecryptionError("Cannot decrypt with a public key")

    @staticmethod
    def int_to_bytes(n: int) -> bytes:
        return int.to_bytes(n, length=n.bit_length(), byteorder=sys.byteorder).rstrip(
            b"\x00"
        )
```

```
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.
    RSAPublicKey) -> bytes:
46          message = int.from_bytes(message, sys.
    byteorder)
47          cipher = pow(message, publicKey.e, publicKey
    .n)
48
49          return RSA.int_to_bytes(cipher)
50
51      @staticmethod
52      def naive_decrypt(ciphertext: bytes, privateKey
    : keys.RSAPrivateKey) -> bytes:
53          # TODO: CRT decryption
54
55          RSA.check_private_key(privateKey)
56
57          ciphertext = int.from_bytes(ciphertext, sys.
    byteorder)
58          plaintext = pow(ciphertext, privateKey.d,
    privateKey.n)
59
60          return RSA.int_to_bytes(plaintext)
61
62      @staticmethod
63      def sign(msg: bytes, key: keys.RSAPrivateKey
    ) -> bytes:
64          # check we have a private key
65
66          # extract integer if sign given bytes
67
68          RSA.check_private_key(key)
69
70          msg = int.from_bytes(msg, sys.byteorder)
71          cipher = pow(msg, key.d, key.n)  # type:
    ignore
```

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

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

```python
# coding=utf-8
"""Interface to all RSA encrypt / decrypt functions
"""

import os
import os.path
import re
import subprocess
from dataclasses import dataclass, field


class RSAKeyLoaderFromNumbers:
    ...


class RSAKeyError(Exception):
    """Problem with RSA Key"""


class RSAParserError(Exception):
    """Error in parsing file containing key"""


class RSAPublicKeyError(Exception):
    ...


@dataclass
class RSAKeyLoader:
    """Will load a key from a private key file"""

    # initialise what we need, don't pass in anything at instantiation
    lookup: dict[str, int] = field(default_factory=lambda: {})
    key: None | str = None

    def load(self, path_to_private_key: str) -> str:
        """
        Loads PEM private key from file in file path;
```

```
38          """
39
40          # will only attempt to load if file exists
41          # if openssl rsa doesn't like file report as
     incorrect format
42
43          try:
44              if not os.path.exists(
    path_to_private_key):
45                  raise RSAParserError(
46                      f"File not found at current path
    : \n{path_to_private_key}"
47                  )
48              key = str(
49                  subprocess.check_output(
50                      f"openssl rsa -noout -text < {
    path_to_private_key}", shell=True
51                  )
52              )
53          except subprocess.CalledProcessError:
54              raise RSAParserError("File not in PEM
    private key format")
55
56          # strip into long easily parsable str; =>
    remove spaces, newlines, colons
57          key = key.replace(" ", "").replace("\\n", ""
    ).replace(":", "")
58
59          # converted from bytes so remove b' ' with
    slice
60          key = key[2:-1]
61
62          # split after title section, discard info
63          # pub exp is given in bin in text format;
    breaks splitting header with )
64          key = re.sub(r"\(0x[0|1]*\)", "", key)  #
    matches of the form (0x[0|1]*)
65          head, key = key.split(")")
66
67          if head != "RSAPrivate-Key(2048bit,2primes":
68              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) ->
   None:
74         """Given loaded SSL info, parses and
   populates n, d, e, p, q"""
75
76         def hexstr_to_int(hexstr: str) -> int:
77             # deals with exception pub exp which is
    always a decimal
78             return int(hexstr) if re.fullmatch("[0-
   9]+", hexstr) else int(hexstr, 16)
79
80         # use local key if key not given
81         if keys is None:
82             keys: str = self.key
83             if self.key is None:
84                 raise RSAParserError("Key has not
   been loaded")
85
86         # set up delimiters; tells us to split when
    parsing
87         delimiters = ["n", "e", "d", "p", "q", "
   exp1", "exp2", "crt_coef"]
88
89         # split into a list along delimiters
90         keys = re.sub(
91             "modulus|publicExponent|privateExponent
   |prime1|prime2|exponent1|exponent2|coefficient",
92             " ",
93             keys,
94         )
95         keys: map = map(hexstr_to_int, keys.split
   ())
96
97         # combine to dict
98         self.lookup = {label: key for label, key in
    zip(delimiters, keys)}
```

```
 99
100
101 @dataclass
102 class RSAPublicKey:
103     def __init__(self, loader: RSAKeyLoader |
    RSAKeyLoaderFromNumbers):
104         self.lookup = loader.lookup
105
106     def __str__(self):
107         return f"n={self.n},\ne={self.e}\n"
108
109     def __getattr__(self, item: str) -> int:
110         """Redefine getattr so that will only give
    n and e"""
111         if item == "n" or item == "e":
112             return self.lookup[item]
113         else:
114             raise RSAPublicKeyError("Requested
    attribute not part of the Public Key")
115
116     def _exists(self, item) -> bool:
117         """Returns an attribute if it exists else
    raise an RSAKeyError"""
118         if self.lookup is not None:
119             try:
120                 return not not self.lookup[item]
121             except KeyError:
122                 raise RSAKeyError(
123                     "Requested attribute not found
    ; have you parsed a key?"
124                 )
125
126
127 @dataclass
128 class RSAKeyLoaderFromNumbers:  # type: ignore
129     lookup: dict[str, int] = field(default_factory=
    lambda: {})
130
131     def load(self, n: int, e: int, d: int = 0) ->
    None:
132         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.
    d}"
147 
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_coef
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 
```

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

```python
# coding=utf-8

"""
Simple RSA implementation for signing / verifying digital signatures.
Will have side channel vulnerabilities due to the way that Python stores numbers
Hashing is done through hashlib library
"""

import hashlib
import sys
from dataclasses import dataclass


class HashError(Exception):
    """Hash provided is not in valid format"""


class HasherError(Exception):
    ...


@dataclass
class Hash:
    def __init__(self, h: bytes):
        self.validate(h)
        self.h: bytes = h

    def int_digest(self) -> int:
        # int representation of bytes, not actual value
        return int.from_bytes(self.h, byteorder=sys.byteorder)

    @staticmethod
    def validate(passed_hash):
        try:
            assert len(passed_hash) == 32  # 256 / 8
            assert type(passed_hash) == bytes
        except AssertionError:
```

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

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

```
 1 -----BEGIN PUBLIC KEY-----
 2 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAqzzsGK6N
   9oxQLtcc4zds
 3 ygDF6Cruv5qlBHlO0rSzQAk+OBy+
   LchMJ02mQKLSw3moxHhogTlJpKecTusBS/PD
 4 /VNYz2gjBLe4Dba4jPyT59Ah40bAHI1z5C/PqH82b51TXO+
   4RqC7MprCdSeeNXQc
 5 L5V+1Ob30GIoyKkw4lCVRrnCclWF5uOE1YnDFjCsdKTW+
   LcwYOQuR7vq2PHpbeWO
 6 m8dJz1Y2zuXTxc27ztqGPcriFrtm7DkV/3Li9g6WsT1AQO0OOF+
   SaiW866+u2zNc
 7 9jST2gKzKSYMktcYPHnusblnRzlCpLpB8hPHEkc2ApdZRvoSc+Mq
   +9kwjUMyXEy/
 8 qwIDAQAB
 9 -----END PUBLIC KEY-----
10
```

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAssGOMnKA
/0q+PtM3sCJR
hWOjNocYURePnuE82Ahch1wIAYTYNsoKPy0ft3HJiTHNviWC/
G6iIJBNTvcznuZz
0UwtkVe8Pq/iZuK976s/Yg+
1SIrTqETTUbwNhmWEL4kjCzI4VEmrjgdnKnZKHDxo
cSnwZUXDv55HNTpAi5CGNwFVQnaKxORDp3nhMY5vnSb0s2vpAEXt
KjKeJydhWLAh
fJP5qEDxD4qOaJQJyZbXsvT+tmm4QnuF+
ptGwDmG3xAZA9Lbtk6xeOplUMzQOKnN
nm9CUBEGoOYAVyGWv9yml/
ucPIaCDdjWOkR0RuMqgqLeiROGjs3BAomkk548Wzdu
FwIDAQAB
-----END PUBLIC KEY-----

```

```
 1 -----BEGIN PUBLIC KEY-----
 2 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwmwTyC9d
   846+33clYUSk
 3 cd+x9i/3j292+vO0wUp1WWA/
   ceJvVbtkyieVAPRmW9o8ow12e67blp0e5TKtksjR
 4 OI7AnVVT2zJgV4Xbfj/Jqu6x5CNdjVA4vwRnYVp+
   hEQv907A2VJZgXTfras0kI6Z
 5 sryHRpGHUs+gjddWfAap/f9dbtSeDi7dPSW+
   Nr6vzwd53N5VadqKd2N2x2CMEUAP
 6 cwYwOn6dGCyojN4E5Mo1/rJ1QEn6y9Hvuqb8OwpudPxw/phKhax+
   VIyDPaH6QMxR
 7 LnZvpepc4HnQrzXmie99Bhb/
   JfAQv34hoNgJ5HnoUzO2n0xTCxelBG7rIWUs/VXG
 8 BQIDAQAB
 9 -----END PUBLIC KEY-----
10
```

```
 1 -----BEGIN RSA PRIVATE KEY-----
 2 MIIEowIBAAKCAQEAqzzsGK6N9oxQLtcc4zdsygDF6Cruv5qlBHlO
   0rSzQAk+OBy+
 3 LchMJ02mQKLSw3moxHhogTlJpKecTusBS/PD/
   VNYz2gjBLe4Dba4jPyT59Ah40bA
 4 HI1z5C/PqH82b51TXO+4RqC7MprCdSeeNXQcL5V+
   1Ob30GIoyKkw4lCVRrnCclWF
 5 5uOE1YnDFjCsdKTW+
   LcwYOQuR7vq2PHpbeWOm8dJz1Y2zuXTxc27ztqGPcriFrtm
 6 7DkV/3Li9g6WsT1AQO0OOF+SaiW866+
   u2zNc9jST2gKzKSYMktcYPHnusblnRzlC
 7 pLpB8hPHEkc2ApdZRvoSc+Mq+9kwjUMyXEy/
   qwIDAQABAoIBAANaLvkQucDA4HT6
 8 Sxt7o0qVF0rDRGdF3MMoqM1hMj0nsSsiDOsSh8MhNwb+
   6Qdgo1gtT0ZwW6u4iEvX
 9 N/
   BHtmeIMS3mSQE3o4fJMvW3oCGipncvlGU6s7Ec6oDO9L7copwoKU
   Bgtyl3dCUd
10 AHDldAPOdmwulV1j/o5nGYjksPdwdVB5w/
   xnt79FfcAqiYQkShLPUSB9vNGvYc3a
11 xLPupa28dW06ae22/XMTvIdMD73AAMfwTXPwIEkYiQ8L/
   rALGgKxiXDvEHoGNUjj
12 8oRJJOI6fflqzWDunf1crs/exPI+wyxCQBsnp9W8R/
   iJjfzgGMgarW7ifz2VeAjl
13 DUhGPlECgYEA4bwY49tNu9D8/wsZHHqEVuMJZWkyQszW/
   9VF2jns8O7feaYkg++H
14 Tax1zIrLjZeyXheEd0ePUCXwp+A+
   puJfMQDUygyd39GJq5JKOQtjLGNfVslcIbws
15 7cveAAKHPgtZ/315DceZp0T/
   etmSIPKfyKfRwXkdT9gc48BJVJCyYbkCgYEAwjJU
16 fBmcife7c7M9iXTgznsptw14l5eWNGUhEtq12ah9B+
   SWO1O2fBvmert17B2qqrfD
17 7WZ+
   Ou5LukVKTZSlYbK3d1tYECi8IX3dTEM16tzI07rldiSLMYmVGB2y
   rUFEla0i
18 GqToI3os5hjgeb2LFnfCDG2ElXyV3cnEKCfhroMCgYEAjlphU+
   gryEKOGYR1LIly
19 xZxdg49oe8nT1/g2Gd2nt3Tch0moA2/
   dYrVcgEYTAdLlUAMCrXn6Pa//aM64k+Nz
20 5mJAzr5QHSJ18DXMCtdMjmSIbiGDsV6K8tc6w8TUZuMfuUf2PCNg
   maSgfGeSiKaY
```

```
21 7yAt3hWzz3NZHKNZWzP42jkCgYA/
   YcntTzamWTLXSnMVQA53lf9BfaYUZCdkJnWq
22 /
   7NGvRVB1DusVWBEFZ8eA70zOWOQoLXT8BXXChuxShg8Rf3Ma1YyI
   KAXdhQhIkFu
23 OKmKZFEvudpWuwqrOmptGpRMs/
   a8m2t8IsKZgbDR0ODDGzggNyoggEY7vBP19XqO
24 WKXyHQKBgEtL9XgELGf+huemb5xcnkhPEPpfhhGLcJdYa0bK/
   zvSvHb5KPW/Nq9O
25 qTTDc1roXW9sXJ+
   Iaz3hUZkCHBLaqFmeqLsXTw9dGIsANjVyxQFiArqLg88H/OP/
26 1zSh3LNMotIryeDXUPhpfaFJurZU3oGC0vqdPBiAFe3KLSsmn3G5
27 -----END RSA PRIVATE KEY-----
28
```

```
 1 -----BEGIN RSA PRIVATE KEY-----
 2 MIIEpAIBAAKCAQEAssGOMnKA/0q+
   PtM3sCJRhWOjNocYURePnuE82Ahch1wIAYTY
 3 NsoKPy0ft3HJiTHNviWC/G6iIJBNTvcznuZz0UwtkVe8Pq/
   iZuK976s/Yg+1SIrT
 4 qETTUbwNhmWEL4kjCzI4VEmrjgdnKnZKHDxocSnwZUXDv55HNTpA
   i5CGNwFVQnaK
 5 xORDp3nhMY5vnSb0s2vpAEXtKjKeJydhWLAhfJP5qEDxD4qOaJQJ
   yZbXsvT+tmm4
 6 QnuF+
   ptGwDmG3xAZA9Lbtk6xeOplUMzQOKnNnm9CUBEGoOYAVyGWv9yml
   /ucPIaC
 7 DdjWOkR0RuMqgqLeiROGjs3BAomkk548WzduFwIDAQABAoIBAAYk
   eIprPNuI6fim
 8 sWcwBrUas2UQR1miAaFoA/s4blWIaA4FTpizYJ7DIi/
   fjRlVyMgb6eBavN95le58
 9 uB1hteEfSQGRjjppONJoQDaYH9lyyLvxaUyC8PACEkToAB7SC446
   74yV63s57eWb
10 ovhNaxlYY5ZXS16uRawikE07biE9r7WVRQIrKesbwMFvDD3T8Vk8
   3Q6m9dAF3uFm
11 En7++v/sEJM1/
   vBFd2QtCGPT9AajqI8XDrvfJVEdIfd8iUP243uwSjgCsGZoJyPq
12 X/beLBK+
   BLtrJB0obw9UUukD23NjVaAd6rg13smasCvd9RN9ak2V1ivRudjM
   kG9S
13 JloBKwECgYEA7JXiTcWXieqvkW9nCdD943Lxj/
   nZSksYj3fpouEEtOgrNNruu6pw
14 igJmNYUA/KUIN1N5pxyOT4x4+
   Wgr8HqcUz1yqQsv9OFtGpVtTWXs8KlBguzFS2/O
15 1zgJayBe/
   7FjT1ZW06wdKw5d8a5ZYaPW0LDsrgDOYlYhTCYht2WQ5ncCgYEAw
   WzP
16 Djnv9JdQdy7Z6DLnOn753Akmjvrf+sKvbsW2HDiqMKZg+
   DTbTlRlsZzCbv6AMe4f
17 Ji+3bA5d+
   Scdaq8BtDP8ZniLXW71KJ8CT0zPoGC02vYuOnnIBFEuprJGH/
   BIKCFZ
18 pYAhowvnN7vDbNrgT+P1T3/pzydb7ZWpWGF4fWECgYB+
   FqO1r0lER7+qUyUnJisp
19 vWjgwtf7sGo2jEIIfMR1Zgcg9E2n1v6DjyPKAki0XYAzfFmppp5x
   5vvScKJ6V95Y
```

```
20 vrcNQNkeOcZcFKdcSKYu3QX9MRM1UTF7onHili4LOA8liW5dAa3J
   9K480B7y2s7y
21 ClkZo3RbFGxKmUf+5jKDoQKBgQCal/
   i1iiD7AKVTXKLVLzoYSrww3S9wP5sctNW6
22 V1NCIxgDYjdGqhhN9q6A0qWkSMz1GzjSMHkNaD47kMO2LMHT4Wju
   DZJ1zuUq3kve
23 Tan0qNZj2zb/ja4LpUb//
   KSHimhhiqY33L0Fr9HRfcSoEBgQQkYikD2zT+Xtchyz
24 gtPmgQKBgQDrs7LmPSk5/
   DvcIrQQTwFR6BXSR7aizJjgbk72taip94KO63Q7RJF3
25 1TWkjDRJdnh5ZPgF+FC5tn23wBK6hj3rF8bNYMJWLAEVoH2/
   Ks0ZSSmQjRPEJIPK
26 GrnOOue1Irs4fl065ptjZtw8HgKitE9sle1vIgwpUKQXyFrfXAit
   kg==
27 -----END RSA PRIVATE KEY-----
28
```

```
 1 -----BEGIN RSA PRIVATE KEY-----
 2 MIIEpQIBAAKCAQEAwmwTyC9d846+33clYUSkcd+x9i/3j292+
   vO0wUp1WWA/ceJv
 3 VbtkyieVAPRmW9o8ow12e67blp0e5TKtksjROI7AnVVT2zJgV4Xb
   fj/Jqu6x5CNd
 4 jVA4vwRnYVp+hEQv907A2VJZgXTfras0kI6ZsryHRpGHUs+
   gjddWfAap/f9dbtSe
 5 Di7dPSW+
   Nr6vzwd53N5VadqKd2N2x2CMEUAPcwYwOn6dGCyojN4E5Mo1/
   rJ1QEn6
 6 y9Hvuqb8OwpudPxw/phKhax+
   VIyDPaH6QMxRLnZvpepc4HnQrzXmie99Bhb/JfAQ
 7 v34hoNgJ5HnoUzO2n0xTCxelBG7rIWUs/
   VXGBQIDAQABAoIBAQCKPl8ykLu1W2LN
 8 cuLZbv2fGvhnnWPiUdfASVnVtyQKES4LxH5173GTbOG+
   dAn0dhF3vzLob0Ukq0Q0
 9 pPxWywQ5uweq1+lwizmGAmA4PRMgv+
   kt1xfGS6yN5ou75aTgV7vjldP6s1uBeaEy
10 0/
   HkeAFB3lwyCXi3oXsAYIXWWsa1G4n2lEj5FlhWLnuJ2KSEA6PejF
   RopB4GzA7m
11 xYyZre4wKN7oTiSke+
   Pm2xYVMzhwxDG0PQdYZQtyle0ZU8n2E8NxAGrh4KkSNVcP
12 9yknoVYz6Lq85lABziMuhmHnnEGs3r2TRkp29aWJ1IQYjX7/
   UFWEBdqrIm9ePkdp
13 vX+
   S6lABAoGBAOOjqqGrAp21sOf2LXh2lsiq5pnGBI2KqLXiO5XfeJM
   HOvN9MWt/
14 tMgDtTeD7gbYC2pXIKpFuyVQjP10Ol/
   R6Hvm3BzaKwEAtXbsfdmzTxqfs7HI8SXR
15 pwKWqr3m6oFOE+
   EuNBhcbfucmhhB3Wg5lseSDHcRxTLIuvjl1bhZzoqBAoGBANqk
16 +32nhDmou11kXMAs7ipv49oc6T6n2cs1Gy/
   Fbqq1lq4TgDDVgoKwznYB1habFy+E
17 e3qCPcu8JVXuOso+
   L0j21yQ1WlC0OlIfObMgqLCpZjzMqa9upDE304ogJqQJDQk2
18 g/
   LG01fQJtafe3QeoHeTZ9wObuk99Qv8guEOtVGFAoGBAMS6pzY8kn
   A1m4N8L2PS
19 KRGXEN+
   IUFr6oD9AGvq1PIDkWMAhS9p5bYUGH1CxBb6Ia6UlUyeUR95BtPc
```

```
19 9wU0a
20 HW8m3sdYjJ27PRhf3YuM+SorJqLY4/8pJsqH51ti+
   vtwvKF4yrDbAHnYpxuboHr4
21 eiRTRc6JXwE6lMuR5ZgClQsBAoGAI8puAJuzYVzljtwm8q5oLjoy
   qjmhVMhVNpZy
22 5NcEzpz7FXPLwDKzMoGOynJygTDSEs01CVDYnMkns3FT3ldflioR
   /bNeHWfjRB4o
23 a9IkywZv3fQCst0Bs6zXy/yHVsLEh4WNA+jYH7/
   LG8bvhoqc6fYPQk56iWPDATtM
24 kVq/A6ECgYEAuXIqXKiiU+xElmI/3F0aMQDk5z+
   Rodrc8hsUM7BehsmrjerHPK1i
25 qzel3EYEzxdt4vLhgHBgpaA1I91duYQcWENDBUZ134wNx7Sf5s4Q
   i/hrlo4obVA6
26 lEwGgQuHE6XQWH10mouAlp4s4snfRreXurfOMYgMEVlGXk0sDAsn
   xm4=
27 -----END RSA PRIVATE KEY-----
28
```

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

```
 1 # coding=utf-8
 2 from setuptools import setup, find_packages  # type
   : ignore
 3
 4 setup(
 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
```

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

```python
# coding=utf-8
from dataclasses import dataclass

import click


@dataclass
class User:
    name: str
    email: str
    modulus: str
    pub_exp: str
    password: str = "default"
    id: int = 0

    def __str__(self):
        return f"""\nName:\t{self.name}
Email:\t{self.email}
Modulus:\t{self.modulus},
Public Exponent:\t{self.pub_exp}
                """


@dataclass
class Group:
    id: int
    name: str
    password: str

    def __repr__(self):
        return f"Group(name={self.name}, id={self.id})"

    def __str__(self):
        return f"Group:\n\tName: {self.name}, ID = {self.id}"


@dataclass
class GroupLink:
    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
    Group: {self.group}", bold=True)
69          click.secho(f"{self.other}, £{(int(self.
    amount) / 100):.2f}", fg="blue")
70          click.secho(self.reference)
71          click.secho(f"at {self.time}")
72
73          if self.verified:
74              click.secho("Verified: True\n", fg="
    green")
75
76          else:
77              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.
    dest_list) else False
 87
 88     def __repr__(self):
 89         return f"{self.src_list}"
 90
 91     def secho(self):
 92         for trn in self.src_list:
 93             trn.secho()
 94         for trn in self.dest_list:
 95             trn.secho()
 96
 97
 98 @dataclass
 99 class Signature:
100     transaction_id: int
101     signature: str  # store as hex
102     origin: str  # src or dest
103
```

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

```python
# coding=utf-8

from marshmallow import Schema, fields, post_load

import models  # type: ignore
import src.transactions.ledger
import src.transactions.transaction


class UserSchema(Schema):
    id = fields.Int()
    name = fields.Str()
    email = fields.Email()
    modulus = fields.Str()
    pub_exp = fields.Str()
    password = fields.Str()

    @post_load
    def make_user(self, data, **kwargs):
        return models.User(**data)


class GroupSchema(Schema):
    id = fields.Int()
    name = fields.Str()
    password = fields.Str()

    @post_load
    def make_group(self, data, **kwargs):
        return models.Group(**data)


class GroupLinkSchema(Schema):
    id = fields.Int()
    group_id = fields.Int()
    usr_id = fields.Int()

    @post_load
    def make_group_link(self, data, **kwargs):
        return models.GroupLink(**data)

```

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

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

```python
# coding=utf-8
import os

import click
from flask import Flask, g
from flask_restful import Resource, Api, abort  # type: ignore

from src.server.processes import get_db
from src.server.resources import (
    Group,
    User,
    UserGroupBridge,
    PrettyTransaction,
    TransactionSigVerif,
    Simplifier,
    GroupDebt,
    SignableTransaction,
)

app = Flask(__name__)
api = Api(app)


@app.teardown_appcontext
def close_connection(exception):
    """Closes db if sudden error"""
    db = getattr(g, "_database", None)
    if db is not None:
        db.close()


@click.group()
def settle_server():
    ...


@click.option("-d", "--debug", is_flag=True, default=False)
@click.option("-h", "--host", default="127.0.0.1")
@settle_server.command()
```

```
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",
   "/transaction/<string:email>")
49
50 api.add_resource(User, "/user/<string:email>", "/
   user")
51
52 api.add_resource(
53     UserGroupBridge, "/group/<int:id>/<string:email
   >", "/group/<string:email>"
54 )
55
56 api.add_resource(
57     TransactionSigVerif, "/transaction/auth/<int:id
   >", "/transaction/auth/"
58 )
59
60 api.add_resource(Simplifier, "/simplify/<int:gid>")
61
62 api.add_resource(GroupDebt, "/group/debt/<int:id>")
63
64 api.add_resource(SignableTransaction, "/transaction/
   settle/<int:t_id>", "/transaction/signable/<int:id>"
   )
65
```

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

```python
# coding=utf-8
import sqlite3

from flask import g

import src.server.models as models
import src.transactions.transaction
from src.crypto import keys as keys

DATABASE = "/home/tcassar/projects/settle/settle_db.sqlite"


class ResourceNotFoundError(Exception):
    ...


def get_db():
    """Returns current database connection"""
    db = getattr(g, "_database", None)
    if db is None:
        # connect
        db = g._database = sqlite3.connect(DATABASE)

    return db


def build_args(data_from_cursor: list | tuple) -> list:
    """Given a ROW OF PARAMETERS FROM DB will return list as args"""
    if args := data_from_cursor:
        args = [item for item in args]
        return args
    else:
        raise ResourceNotFoundError(
            "Database error: failed to build schema of object as nothing was retrieved"
        )


```

```
38 def build_pretty_transactions(
39     src_sql: str, dest_sql: str, cursor: sqlite3.
   Cursor, args: list
40 ) -> models.PrettyList:
41     """Returns pretty list of unchecked transactions
   """
42     # TODO: add transaction verification
43 
44     if type(args) is not list:
45         args = list(args)
46 
47     src_transactions: list[models.PrettyTransaction
   ] = []
48     dest_transactions: list[models.PrettyTransaction
   ] = []
49 
50     src_data = cursor.execute(src_sql, args)
51 
52     for row in src_data.fetchall():
53         pretty = models.PrettyTransaction(*
   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(*
   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,
   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
```

```
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), src_key_data)
119         dest_key_data = map(lambda x: int(x, 16), dest_key_data)
120 
121         # load to key loaders
122         src_ldr = keys.RSAKeyLoaderFromNumbers()
123         dest_ldr = keys.RSAKeyLoaderFromNumbers()
124 
125         src_ldr.load(*src_key_data)  # type: ignore
126         dest_ldr.load(*dest_key_data)
127 
128         # add keys to transaction data in the right place so that they can be unpacked as positional arguments
129         transaction_data.insert(3, dest_ldr.pub_key())
130         transaction_data.insert(3, src_ldr.pub_key())
131 
132         # build signatures dict for transaction
133 
134         signatures = {}
135 
136         for sig, notary in zip([src_sig, dest_sig], [src_id, dest_id]):
137             if sig:
138                 signatures[notary] = int(sig, 16)
139 
```

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

```
174
175 def verify_pretty(
176     pretty: models.PrettyTransaction, cursor:
    sqlite3.Cursor
177 ) -> models.PrettyTransaction:
178     """Update the verification status of pretty
    depending on signatures"""
179     try:
180         t = get_verified_transaction_by_id(pretty.
    id, cursor)
181         t.verify()
182         pretty.verified = True
183     except src.transactions.transaction.
    VerificationError:
184         pretty.verified = False
185         print(f"Signature of {pretty.id} invalid")
186
187     return pretty
188
189
190 def push_transaction(
191     transaction: src.transactions.transaction.
    Transaction, cursor: sqlite3.Cursor
192 ):
193
194     # check pair exists; append if not
195     pair_id: int = cursor.execute(
196         """SELECT pairs.id FROM pairs WHERE src_id
     = ? and dest_id = ?""",
197         [
198             transaction.src,
199             transaction.dest,
200         ],
201     ).fetchone()[0]
202
203     if not pair_id:
204         cursor.execute(
205             """INSERT INTO pairs (src_id, dest_id)
    VALUES (?, ?)""",
206             [
207                 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
    .key_id
217                         JOIN users u2 on keys.id =
    u2.key_id
218                         WHERE u.id = ? OR u2.id
     = ?""",
219         [transaction.src, transaction.dest],
220     ).fetchall()
221 
222     sql = """INSERT INTO transactions
223     (pair_id, group_id, amount, src_key, dest_key,
    reference, time_of_creation)
224             VALUES (?, ?, ?, ?, ?, ?, ?)"""
225 
226     print(key_ids[0][0], key_ids[1][0])
227 
228     # append unsigned transactions
229     cursor.execute(
230         sql,
231         [
232             pair_id,
233             transaction.group,
234             transaction.amount,
235             key_ids[0][0],
236             key_ids[1][0],
237             transaction.reference,
238             transaction.time
239         ],
240     )
241 
242     get_db().commit()
243 
```

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

```python
# coding=utf-8
import json

from flask import request
from flask_restful import Resource, abort  # type: ignore

import src.transactions.transaction as transactions
import src.transactions.ledger as ledgers
from src.server import models as models, schemas as schemas, processes as processes


# Resources


class Group(Resource):
    def get(self, id: int):
        cursor = processes.get_db().cursor()
        # check group exists and
        get_group = """SELECT id, name, password FROM groups
                    WHERE groups.id = ?"""
        try:
            group_data = cursor.execute(get_group, [id])
            group_data = group_data.fetchall()
            # create group object
            group = models.Group(*processes.build_args(*group_data))
        except IndexError:
            abort(404, message="Group ID does not exist")
            group = ""  # type: ignore
        except TypeError as te:
            abort(404, message=f"Group data invalid ; {te}")
            group = ""  # type: ignore
        # create group schema
        schema = schemas.GroupSchema()
        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)
    VALUES (?, ?)""",
46              [group.name, group.password],
47          )
48
49          processes.get_db().commit()
50
51          return f"Created group ID={cursor.lastrowid}
     named {group.name}", 201
52
53
54  class User(Resource):
55      def get(self, email: str):
56          # query db for all users
57          cursor = processes.get_db().cursor()
58
59          query = """
60                    SELECT users.name, users.email,
    keys.n, keys.e, users.password, users.id
61                    FROM users, keys
62                    WHERE email = ? AND keys.id = users
    .key_id;
63                  """
64
65          # only return one usr, so unpack first into
    usr_data
66          try:
67              usr_data: list = cursor.execute(query, [
    email]).fetchone()
68          except IndexError:
69              # 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
    usr_data])
78              schema = schemas.UserSchema()
79          except TypeError:
80              # didn't have required arguments to
    build usr
81              return "User data invalid", 404
82
83          return schema.dump(usr), 200
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
     email = ?
94          """
95
96          exists = cursor.execute(query, [usr.email])
97          if exists.fetchall():
98              abort(409, message="User already exists
    ")
99
100         # add user to db
101
102         keys_query = """INSERT INTO keys (n, e)
103                         VALUES (?, ?)"""
104
105         users_query = """INSERT INTO users (NAME,
    EMAIL, PASSWORD, KEY_ID)
```

```
106                             VALUES (?, ?, ?, ?)"""
107
108         cursor.execute(keys_query, [usr.modulus,
    usr.pub_exp])
109
110         key_id = cursor.lastrowid
111         cursor.execute(users_query, [usr.name, usr.
    email, usr.password, key_id])
112         processes.get_db().commit()
113
114         return (schema.dump(usr),)
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
    they have been validated by client already
124         uid_sql = """SELECT id FROM users WHERE
    email = ?"""
125
126         glink_sql = """INSERT INTO group_link (
    group_id, usr_id)
127                             VALUES (?, ?)"""  # group
    id then user id
128
129         cursor = processes.get_db().cursor()
130         uid = cursor.execute(uid_sql, [email]).
    fetchone()[0]
131
132         cursor.execute(glink_sql, [id, uid])
133
134         processes.get_db().commit()
135
136         glink_data = cursor.execute(
137             """SELECT * from group_link WHERE id
     = ?""", [cursor.lastrowid]
138         )
```

```
139
140         try:
141             print("making group")
142             glink = models.GroupLink(*processes.
    build_args(glink_data.fetchone()))
143         except processes.ResourceNotFoundError as
    re:
144             return 404, f"{re}, failed"
145
146         schema = schemas.GroupLinkSchema()
147
148         return schema.dump(glink), 201
149
150     def get(self, email: str):
151         """Return all groups that a user is part of
    """
152
153         sql = """SELECT g.id, g.name, g.password
    FROM users
154         JOIN group_link gl on users.id = gl.usr_id
155         JOIN groups g on g.id = gl.group_id
156         WHERE users.id
157                   = (
158             SELECT users.id FROM users
159             WHERE email = ?
160             );
161         """
162
163         cursor = processes.get_db().cursor()
164         group_data = cursor.execute(sql, [email])
165         groups: list[models.Group] = []
166
167         for row in group_data.fetchall():
168             groups.append(models.Group(*processes.
    build_args(row)))
169
170         groups_obj = models.GroupList(groups)
171         groups_schema = schemas.GroupListSchema()
172
173         return groups_schema.dump(groups_obj), 200
174
```

```
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 = """SELECT transactions.id,
    group_id, amount, reference, time_of_creation, u2.
    email FROM transactions
187                 INNER JOIN pairs p on p.id =
    transactions.pair_id
188                 INNER JOIN users u on u.id = p.
    src_id
189                 INNER JOIN users u2 on u2.id = p.
    dest_id
190                 WHERE transactions.src_settled = 0
    AND u.email = ?
191         """
192
193         dest_sql = """
194         SELECT transactions.id, group_id, amount,
    reference, time_of_creation, u2.email FROM
    transactions
195                 INNER JOIN pairs p on p.id =
    transactions.pair_id
196                 INNER JOIN users u on u.id = p.
    dest_id
197                 INNER JOIN users u2 on u2.id = p.
    src_id
198                 WHERE transactions.src_settled = 0
    AND u.email = ?"""
199
200         pretty_list = processes.
    build_pretty_transactions(
```

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

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

```
266                 ],
267             )
268 
269             processes.get_db().commit()
270 
271             return cursor.lastrowid, 201
272 
273 
274 class TransactionSigVerif(Resource):
275     def get(self, id):
276         """Verify a transaction, returning pretty
    copy of verified transaction"""
277 
278         try:
279             transaction = processes.
    get_verified_transaction_by_id(
280                 id, processes.get_db().cursor()
281             )
282         except processes.ResourceNotFoundError as
    rnfe:
283             return str(rnfe), 404
284 
285         try:
286             transaction.verify()
287             verified = True
288         except transactions.VerificationError:
289             verified = False
290 
291         # get emails involved in transaction, check
     if user is src or dest
292         emails = (
293             processes.get_db()
294             .cursor()
295             .execute(
296                 """
297                     SELECT u.email, u2.email FROM
    transactions
298                     JOIN pairs p on transactions.
    pair_id = p.id
299                     JOIN users u on p.src_id = u.id
300                     JOIN users u2 on dest_id = u2.
```

```
300 id
301                         WHERE transactions.id = ?
302                     """,
303                     [id],
304                 )
305                 .fetchone()
306             )
307 
308             # build pretty transaction to send back to
    the user
309 
310             pretty = processes.transaction_to_pretty(
    emails, transaction, verified)
311 
312             schema = schemas.PrettyTransactionSchema()
313 
314             return schema.dump(pretty), 200
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
    one of the parties in this transaction",
335                     403,
336                 )
337 
```

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

```
371
372         # simplify debt system
373         try:
374             ledger.simplify_ledger()
375         except ledgers.NoFutherSimplifications:
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,
    amount, time_of_creation, reference, u.email, u2.
    email
409                     FROM transactions
410                     INNER JOIN pairs p on p.id =
    transactions.pair_id
411                     INNER JOIN users u on u.id = p.
    src_id
412                     INNER JOIN users u2 on u2.id =
    p.dest_id
413                     WHERE group_id = ? AND (
    transactions.src_settled = 0 OR transactions.
    dest_settled = 0);
414             """
415
416         trns: list[models.PrettyTransaction] = []
417         for row in cursor.execute(sql, [id]).
    fetchall():
418             row = list(row)
419             dest = row.pop()
420             src = row.pop()
421             row.append(f"{src} -> {dest}")
422
423             pretty = models.PrettyTransaction(*
    processes.build_args(row), False)
424             pretty = processes.verify_pretty(pretty
    , cursor)
425
426             trns.append(pretty)
427
428         schema = schemas.PrettyListSchema()
429         return schema.dump(models.PrettyList(trns
    , [])), 200
430
431
432 class SignableTransaction(Resource):
433     def get(self, id):
434         """Returns a fully signable transaction
    object"""
435
436         try:
```

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

```python
# coding=utf-8
import logging
import sys
from dataclasses import dataclass
from typing import Callable

from ordered_set import OrderedSet

# initialise logger
import src.simplify.graph_objects
from src.simplify import base_graph as graphs

logging.basicConfig(stream=sys.stdout, encoding="utf-8", level=logging.DEBUG)


# typing
disc_map = dict[
    src.simplify.graph_objects.Vertex, src.simplify.graph_objects.Vertex | bool
]
"""disc_map used to track which nodes have been discovered; {node: discovered?}"""


prev_map = dict[
    src.simplify.graph_objects.Vertex, src.simplify.graph_objects.Vertex | None
]
"""prev_map used to track a path through a graph; stored as {node: comes_from_node}.
If node has no links (i.e. is start node), value is None"""


class SearchError(Exception):
    def __init__(self, text, node: src.simplify.graph_objects.Vertex):
        super(SearchError, self).__init__(text, node)
        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
   functionality such as a maxflow along each edge
   during a BFS"""
41     pass
42
43
44 def str_map(generic_map: disc_map | prev_map) -> str
   :
45     """""""
46     str_rep = ""
47     for node, prev in generic_map.items():
48         str_rep += f"{node}: {prev},\n".upper()
49     return str_rep
50
51
52 @dataclass(init=False)
53 class BFSQueue:
54     def __init__(self, *args: src.simplify.
   graph_objects.Vertex):
55         self.data: OrderedSet[src.simplify.
   graph_objects.Vertex] = OrderedSet(args)
56
57     def __str__(self):
58         str_ = ""
59         for datum in self.data:
60             str_ += str(datum).upper()
61         return str_
62
63     def enqueue(self, *args: src.simplify.
   graph_objects.Vertex):
64         for v in args:
65             graphs.GenericDigraph.sanitize(v)
66             self.data.append(v)
67
68     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 walking through graph"""
 77
 78     """Shortest path code: returns list[Vertex], hops along a path"""
 79
 80     @staticmethod
 81     def build_bfs_structs(
 82         graph: graphs.GenericDigraph,
 83         src: None | src.simplify.graph_objects.Vertex = None,
 84     ) -> tuple[BFSQueue, disc_map, prev_map]:
 85         """Helper function to initialise prev_map, disc_map and bfs queue;
 86         if source is passed then queue initialised with src"""
 87         queue = BFSQueue()
 88         disc: disc_map = {node: False for node in graph.nodes()}
 89         prev: prev_map = {node: None for node in graph.nodes()}
 90
 91         if src:
 92             queue.enqueue(src)
 93         else:
 94             # get 'first' item from graph
 95             # n = len(graph.graph)
 96             # start = list(graph.graph.keys())[random.randint(0, n - 1)]
 97             # queue.enqueue(start)
 98             start = next(iter(graph.graph))
 99             logging.debug(f"starting from {start}")
100
101             # queue.enqueue(next(iter(graph.graph
```

```
101 )))
102 
103         return queue, disc, prev
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
    find path between nodes
114         Accepts graph, source node, sink node,
    returns list of nodes, which is the path from src
    to sink
115         """
116 
117         # create queue, discovered list, previous
    list
118         queue, discovered, prev = Path.
    build_bfs_structs(graph, source)
119 
120         # recursive call
121         previous = Path.BFS(
122             graph=graph,
123             queue=queue,
124             discovered=discovered,
125             target=sink,
126             previous=prev,
127             neighbours=neighbours,
128         )
129 
130         return Path._build_path(previous, sink)
131 
132     @staticmethod
133     def _build_path(
134         previous: prev_map, sink: src.simplify.
    graph_objects.Vertex
135     ) -> list[src.simplify.graph_objects.Vertex]:
```

```
136         """Given a mapping of previous nodes,
    reconstructs a path to sink"""
137         path: list[src.simplify.graph_objects.
    Vertex] = [sink]
138 
139         while current := previous[path[0]]:
140             path.insert(0, current)
141 
142         if path[0] == sink:
143             path = []
144 
145         return path
146 
147     @staticmethod
148     def BFS(
149         *,
150         graph: graphs.GenericDigraph,
151         queue: BFSQueue,
152         discovered: disc_map,
153         target: src.simplify.graph_objects.Vertex
     | None,
154         previous: prev_map,
155         neighbours: Callable,
156         do_to_neighbour: Callable = void,
157     ) -> prev_map:
158         """BFS Search as part of finding the
    shortest path through unweighted graph from src ->
    target.
159         Target = None => walk through entire graph
    , terminate at empty queue
160         Returns 'previous' list so that path can be
     rebuilt.
161         Can pass in a function `do_to_neighbour` to
     do to all nodes during the BFS
162 
163         neighbour function need to return edge
164         """
165         # breakpoint: f"q: {queue}\n discovered: {
    str_map(discovered)}\nprev: {str_map(prev)}"
166 
167         # 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
    standalone node (i.e. no forward or backwards links
    )
177             if not graph.connected(current):
178                 if not queue:
179                     raise SearchError("Cannot
    traverse a non connected node", current)
180 
181             # if discovered target node return prev
182             if current == target:
183                 return previous
184 
185             else:
186                 # otherwise, continue on
187                 # enqueue neighbours, keep track of
     whose neighbours they are given not already
    discovered
188                 # do passed in function to
    neighbouring nodes
189                 for neighbour in neighbours(current
    ):
190                     if not discovered[neighbour.
    node]:
191                         previous[neighbour.node] =
    current
192                         queue.enqueue(neighbour.
    node)
193 
194                     do_to_neighbour(current,
    neighbour.node)
195 
196                 # recursive call on new state
197                 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
```

```python
# coding=utf-8

"""
Set up graph object to be used in condensing debt settling
"""

import copy

import graphviz

from src.simplify.graph_objects import Vertex, Edge


class GraphError(Exception):
    ...


class Default:
    def __gt__(self, other):
        return True


class GenericDigraph:
    def __init__(self, vertices: list[Vertex]) -> None:
        """
        Sets up a graph given a list of vertices
        """
        # initialise with values being empty
        # build dict checking each type as we go

        self.graph: dict[Vertex, list[Edge]] = {
            vertex: [] if self.sanitize(vertex) else None  # type: ignore
            for vertex in vertices
        }

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

```
 75     @staticmethod
 76     def edge_from_nodes(node: Vertex, list_: list[
    Edge]) -> Edge:
 77         """Checks if node is in a list of edges,
    will return relevant edge if found
 78         usage:"""
 79         for edge in list_:
 80             if edge.node == node:
 81                 return edge
 82             else:
 83                 continue
 84 
 85         raise GraphError("Node not in list")
 86 
 87     @staticmethod
 88     def nodes_from_edges(edges: list[Edge]) -> list
    [Vertex]:
 89         nodes = []
 90         for edge in edges:
 91             nodes.append(edge.node)
 92         return nodes
 93 
 94     @staticmethod
 95     def sanitize(v: Vertex, *args) -> bool:
 96         """Raises GraphGenError if args are not
    Vertex"""
 97         if args is not None:
 98             tests = [v, *args]
 99         else:
100             tests = [v]
101         for test in tests:
102             if type(test) is not Vertex:
103                 raise GraphError(f"{test} if of
    type {type(test)} not Vertex ")
104 
105         return True
106 
107     def is_node(self, v: Vertex) -> bool:
108         return v in self.graph
109 
110     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,
    list[Edge]]:
115         """Pops node, returns key/value pair of
    node and previous connections"""
116         # look at _backwards_graph to find
    associations
117         for edge in self._backwards_graph[v]:
118             # removing B from A and C; A's pass
119             pointing_node_neighbours = self.
    _backwards_graph[
120                 edge.node
121             ]  # [Edge(A), Edge(C)]
122             pointing_edge = self.edge_from_nodes(v
    , self.graph[edge.node])
123             self.graph[edge.node].remove(
    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 (
    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) ->
    Edge:
136         self.sanitize(s, t)
137         edge = self.edge_from_nodes(t, self.graph[s
    ])
138 
139         if edge is None:
140             raise GraphError("Cannot pop edge that
    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
    False
154 
155 
156 class Digraph(GenericDigraph):
157     def add_edge(self, s: Vertex, *args: Vertex
    ) -> None:
158         self.sanitize(s, *args)
159         for target in args:
160             self.graph[s].append(Edge(target))
161             self._backwards_graph[target].append(
    Edge(s))
162 
```

```python
# coding=utf-8
from dataclasses import dataclass

from src.simplify.base_graph import GenericDigraph, GraphError
from src.simplify.graph_objects import Vertex

"""Flow Graph"""


class FlowEdgeError(Exception):
    ...


class EdgeCapacityZero(Exception):
    ...


@dataclass(init=False, repr=False, eq=False)
class FlowEdge:
    def __init__(self, src: Vertex, node: Vertex, capacity: int, flow: int = 0):
        self.src: Vertex = src
        self.node: Vertex = node  # where is edge pointing
        self.capacity: int = capacity  # non -ve;
        self.flow: int = flow  # always initialised to 0
        self.residual: bool = not capacity

    def __str__(self):
        return (
            f"{self.node} [{self.flow}/{self.capacity}], "
            if not self.residual
            else f"R: {self.node} [{self.flow}/{self.capacity}],"
        )

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

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

```
134                 self[dest].append(FlowEdge(dest,
    src, 0))
135
136                 self[src].sort(reverse=True)
137
138             # edge going in direction of edge being
     added
139             elif self.is_edge(src, dest):
140                 self.get_edge(src, dest).capacity
     += capacity
141
142             elif self.is_edge(dest, src, residual=
    False):
143                 new_cap = self.get_edge(dest, src).
    capacity - capacity
144                 # if new capacity is 0 pop edge
145
146                 if new_cap > 0:
147                     self.get_edge(dest, src).
    capacity = new_cap
148                 if new_cap < 0:
149                     self.pop_edge(dest, src)
150                     self.add_edge(src, (dest,
    new_cap * -1))
151                 if new_cap == 0:
152                     self.pop_edge(dest, src)
153
154
155             if update_debt:
156                 # handle net_debt;
157                 self.net_debt[src] += capacity
158                 self.net_debt[dest] -= capacity
159
160     def pop_edge(self, src: Vertex, dest: Vertex
    , *, update_debt=False):
161         """removes REAL edges, and deletes residual
     counterpart"""
162
163         # normal
164         fwd_edge = self.edge_from_nodes(dest, self[
    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 ede + 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
     my graph
201                     # delete edge + residual; will
    condition only ever raised on fwd edges
202                     self.pop_edge(node, edge.node)
203
204     #
205     # def nodes(self):
206     #     """Returns nodes in order of incoming
    edges (smallest first)"""
207     #     def incoming_func(node):
208     #         return len([edge for edge in self[
    node] if edge.residual])
209     #
210     #     incoming = [(node, incoming_func(node))
    for node in self.graph.keys()]
211     #     incoming.sort(key=lambda row: row[1],
    reverse=True)
212     #     return [node for node, _ in incoming
213
```

```python
# coding=utf-8

from dataclasses import dataclass


"""All objects to be used in various graphs"""


class FlowError(Exception):
    ...


@dataclass
class Vertex:
    """Representation of a vertex; carries data and ID"""

    ID: int  # IDs should be unique
    label: str = ""  # optional label

    def _key(self) -> tuple:
        """Returns immutable repr of object for hashing"""
        return self.ID, self.label

    def __str__(self):
        return self.label if self.label else str(self.ID)

    def __hash__(self):
        """for adding to lists / dicts"""
        return hash(bytes(f"{self._key()}".encode("utf8")))


@dataclass
class Edge:
    node: Vertex

    def __str__(self):
        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
     if capacity is 0
60
61      def __str__(self):
62          return (
63              f"{self.node} [{self.flow}/{self.
    capacity}], " if not self.residual else ""
64          )
65
66      def to_dot(self):
67          base = f'[label="{self.flow}/{self.capacity}
    "]'
68          return base[:-1] + ", color=red]" if self.
    residual else base
69
70      def unused_capacity(self) -> int:
71          return self.capacity - self.flow
72
73      def push_flow(self, flow):
74          if self.flow + flow > self.capacity or type(
```

```
flow) is not int:
            raise FlowError(
                f"Cannot send {flow} units down path of capacity {self.capacity}"
            )
        self.flow += flow

```

```python
# coding=utf-8

import copy

from src.simplify import path as path
from src.simplify.flow_graph import FlowGraph, FlowEdge
from src.simplify.graph_objects import Vertex


class SettleError(Exception):
    ...


class NoOptimisations(Exception):
    """No optimisations to graph; already in simplest form"""


class MaxFlow:
    @staticmethod
    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)
            # graph.to_dot()

        return max_flow

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

```
35          """find the shortest path from src -> sink
    using BFS"""
36          return path.Path.shortest_path(
37              graph, src, sink, neighbours=graph.
    flow_neighbours
38          )
39
40      @staticmethod
41      def bottleneck(graph: FlowGraph, node_path: list
    [Vertex]) -> int:
42          """Returns bottleneck of a path"""
43          aug_path = MaxFlow.nodes_to_path(graph,
    node_path)
44          remaining = [edge.unused_capacity() for edge
     in aug_path]
45          return min(remaining)
46
47      @staticmethod
48      def augment_flow(graph: FlowGraph, node_path:
    list[Vertex], flow: int) -> None:
49          """Adds flow to normal edges of path,
    subtracts from residual
50          deals with pushing to residual and hence
    subtracting from normal"""
51          # normal edges
52          aug_path = MaxFlow.nodes_to_path(graph,
    node_path)
53          for edge in aug_path:
54              edge.push_flow(flow)
55
56          # flip node path to give residual
57          node_path.reverse()
58          # get edges from reversed path
59          res_path = MaxFlow.nodes_to_path(graph,
    node_path)
60
61          # push flow down res path
62          for edge in res_path:
63              edge.push_flow(flow * -1)
64
65      @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 not debt:
92              edge: FlowEdge  # type: ignore
93              for (node, adj_list) in debt.graph.
    items():
94
95                  for edge in adj_list:  # type:
    ignore
96                      if not edge.residual:
97                          if flow := MaxFlow.
    edmonds_karp(debt, node, edge.node):
98                              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
```

```python
# coding=utf-8

from src.simplify.base_graph import GenericDigraph, GraphError
from src.simplify.graph_objects import Vertex, WeightedEdge


class WeightedDigraph(GenericDigraph):
    def add_edge(self, source: Vertex, *edges: tuple[Vertex, int]) -> None:

        # sanitize source
        self.sanitize(source)
        for node, weight in edges:
            if self.is_edge(source, node):
                existing: WeightedEdge
                existing = self.edge_from_nodes(node, self[source])  # type: ignore
                existing.weight += weight
            else:
                self.sanitize(node)
                self.graph[source].append(WeightedEdge(node, weight))
                self._backwards_graph[node].append(WeightedEdge(source, weight * -1))

    def flow_through(self, node: Vertex) -> int:
        """Returns sum of weights out of node
        +ve => net flow out of node
        -ve => net flow into node"""
        flow = 0
        edge: WeightedEdge

        for graph in [self.graph, self._backwards_graph]:
            for edge in graph[node]:  # type: ignore
                flow += edge.weight

        # return -ve, as backwards edges have -ve 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
    owed (-ve if they need to pay)"""
38          return {node: self.flow_through(node) for
    node in self.nodes()}
39
40      def is_edge(self, s: Vertex, t: Vertex) -> int:
41          try:
42              edge: WeightedEdge
43              edge = self.edge_from_nodes(t, self.
    graph[s])  # type: ignore
44              return edge.weight
45          except GraphError:
46              return 0
47
```

```python
# coding=utf-8

import csv
import os
from dataclasses import dataclass, field

import src.simplify.flow_algorithms
import src.simplify.flow_graph as flow
from src.crypto import keys
from src.simplify.flow_algorithms import Simplify, SettleError
from src.simplify.graph_objects import Vertex
from src.transactions.transaction import Transaction, VerificationError


class LedgerBuildError(Exception):
    """Error building ledger"""


class NoFutherSimplifications(Exception):
    ...


@dataclass
class Ledger:
    """Multiple transactions contained to one group (assumed from building);
    built from a stream of transaction objects"""

    # ledger, big list of transactions;
    # TODO: maybe make ledger generator
    ledger: list[Transaction] = field(default_factory=lambda: [])
    nodes: list[Vertex] = field(default_factory=lambda: [])
    key_map: dict[int, keys.RSAPublicKey] = field(default_factory=lambda: {})

    def __bool__(self):
        """False if ledger empty"""
```

```
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) ->
    list[Transaction]:
42          """Nice syntax for adding transactions to
    ledger"""
43
44          if type(transaction) is not Transaction:
45              raise LedgerBuildError(
46                  f"cannot append type {transaction}
    to ledger; must be transaction"
47              )
48          else:
49              self.ledger.append(transaction)
50              self.key_map[transaction.src] =
    transaction.src_pub
51              self.key_map[transaction.dest] =
    transaction.dest_pub
52
53          return self.ledger
54
55      def _verify_transactions(self) -> None:
56          """Verifies the keys of all the transactions
     in the group.
57          Raises error if a faulty transaction is
    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), (
    Vertex(trn.dest), trn.amount))
 80 
 81         return as_flow
 82 
 83     def _flow_to_transactions(self, fg: flow.
    FlowGraph) -> list[Transaction]:
 84         """For each edge, make a transaction"""
 85 
 86         new_trns: list[Transaction] = []
 87 
 88         edge: flow.FlowEdge
 89 
 90         # go through every outgoing transaction by
    person
 91         for person, adj_list in fg.graph.items():
 92             for edge in adj_list:  # type: ignore
 93 
 94                 # don't add residual edges to
    ledger
 95                 if edge.residual:
 96                     continue
 97 
 98                 else:
 99                     # generate UNSIGNED
    transactions
100                     # TODO: let db handle id; keep
    it initialised to 0
101                     edge: flow.FlowEdge  # type:
    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(
    fg)
120
121             # settle, update ledger
122             simplified_fg.to_dot(title="settled")
123             self.ledger = self.
    _flow_to_transactions(simplified_fg)
124
125         except SettleError:
126             # no changes made to graph, keep ledger
     as is, with sigs.
127             # print(
128             #     "Graph already at few
    transactions per person; no optimisations found"
129             # )
130             raise SettleError('Failed to settle
    graph... aborted')
131
132         except src.simplify.flow_algorithms.
    NoOptimisations:
133             raise NoFutherSimplifications(
134                 "Graph already at few transactions
    per person; no optimisations found"
135             )
136
137         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.
    RSAPublicKey, int
154         ]:
155 
156             ldr = keys.RSAKeyLoaderFromNumbers()
157             ldr.load(n=int(row[get_field("src_n"
    )]), e=int(row[get_field("src_e")]))  # type:
    ignore
158             src_pub: keys.RSAPublicKey = keys.
    RSAPublicKey(ldr)
159 
160             ldr2 = keys.RSAKeyLoaderFromNumbers()
161             ldr2.load(n=int(row[get_field("dest_n"
    )]), e=int(row[get_field("dest_e")]))  # type:
    ignore
162             dest_pub: keys.RSAPublicKey = keys.
    RSAPublicKey(ldr2)
163 
164             print(src_pub, dest_pub, "---", sep="\n
    ")
165 
166             return (
167                 int(row[get_field("src")]),
168                 int(row[get_field("dest")]),
169                 int(row[get_field("amount")]),
```

```
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: \
    n{os.getcwd()};\nsearched for {path}"
178             )
179 
180         transactions: list[list[Transaction]] = []
181 
182         # generate transaction objects, store as
    list of groups of transactions
183         with open(path) as csvfile:
184             transaction_reader = csv.reader(csvfile
    , delimiter=",")
185             for row in transaction_reader:
186                 # use header to build index of
    where things are
187                 if row[0] == "ID":
188                     header: list[str] = row
189                     continue
190 
191                 # build transaction, keep groups
    intact
192 
193                 trn = Transaction(*build_trn())
194 
195                 try:
196                     transactions[int(row[get_field(
    "group")])].append(trn)
197 
198                 except IndexError:
199                     # make position at group if not
     made yet (assuming consecutive 0 indexed group
    numbers
200                     transactions.append([trn])
201 
202         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
```

```python
# coding=utf-8

"""
Handles transaction object
"""

import datetime

# coding=utf-8
import sys
from abc import ABC, abstractmethod
from dataclasses import dataclass, field

from src.crypto import keys, hashes, rsa


class TransactionError(Exception):
    ...


class VerificationError(Exception):
    ...


class Signable(ABC):
    """Base class for objects that can be signed; needs a hash implementation, cannot be added into ABC for reasons"""

    @abstractmethod
    def sign(self, key: keys.RSAPrivateKey, *, origin: str) -> None:
        ...

    @abstractmethod
    def hash(self) -> bytes:
        ...


@dataclass
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(
    default_factory=lambda: {})
48      group: int = 0
49
50      def hash(self) -> bytes:
51          # standard way to produce hash using SHA256
    interface from crypto lib
52          return (
53              hashes.Hasher(
54                  f"{self.src, self.dest, self.amount
    , self.reference, self.time}".encode(
55                      "utf8", sys.byteorder
56                  )
57              )
58              .digest()
59              .h
60          )
61
62      # note: IMPORTANT: CANNOT USE DUNDER HASH AS
    PYTHON DOES WEIRD HASH COMPRESSION
63      def __hash__(self):
64          raise NotImplementedError("__hash__ cannot
    be used, used .hash() method")
65
66      def sign(self, key: keys.RSAPrivateKey, *,
    origin: str) -> None:
67          def overwrite_err(where: str) -> None:
68              """Helper to raise overwrite error"""
69              raise TransactionError(f"Cannot
    overwrite signature of {where}")
70
71          def generate_signature(obj: Transaction) ->
```

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

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

```
135 16).to_bytes(256, sys.byteorder)
136
137         for key, origin in zip(usr_keys, [self.src
    , self.dest]):
138             hash_from_sig = rsa.RSA.inv_sig(self.
    signatures[origin], key)
139             if hash_from_sig != hash_from_obj:
140                 raise VerificationError(f"{self.src
    } sig invalid")
141
```

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 — 600 ms
  - test_hashes — 11 ms
    - TestHash — 11 ms
      - test_hash (tests that hash object can validate hash looking numbers) — 9 ms
        - [valid] — passed
        - [too short] — passed
        - [too long] — passed
        - [wrong type] — 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 — 377 ms
    - TestRSAKeyLoading — 377 ms
      - 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 — 60 ms
        - [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']

Collapse | Expand

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

test_add_edge passed 0 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

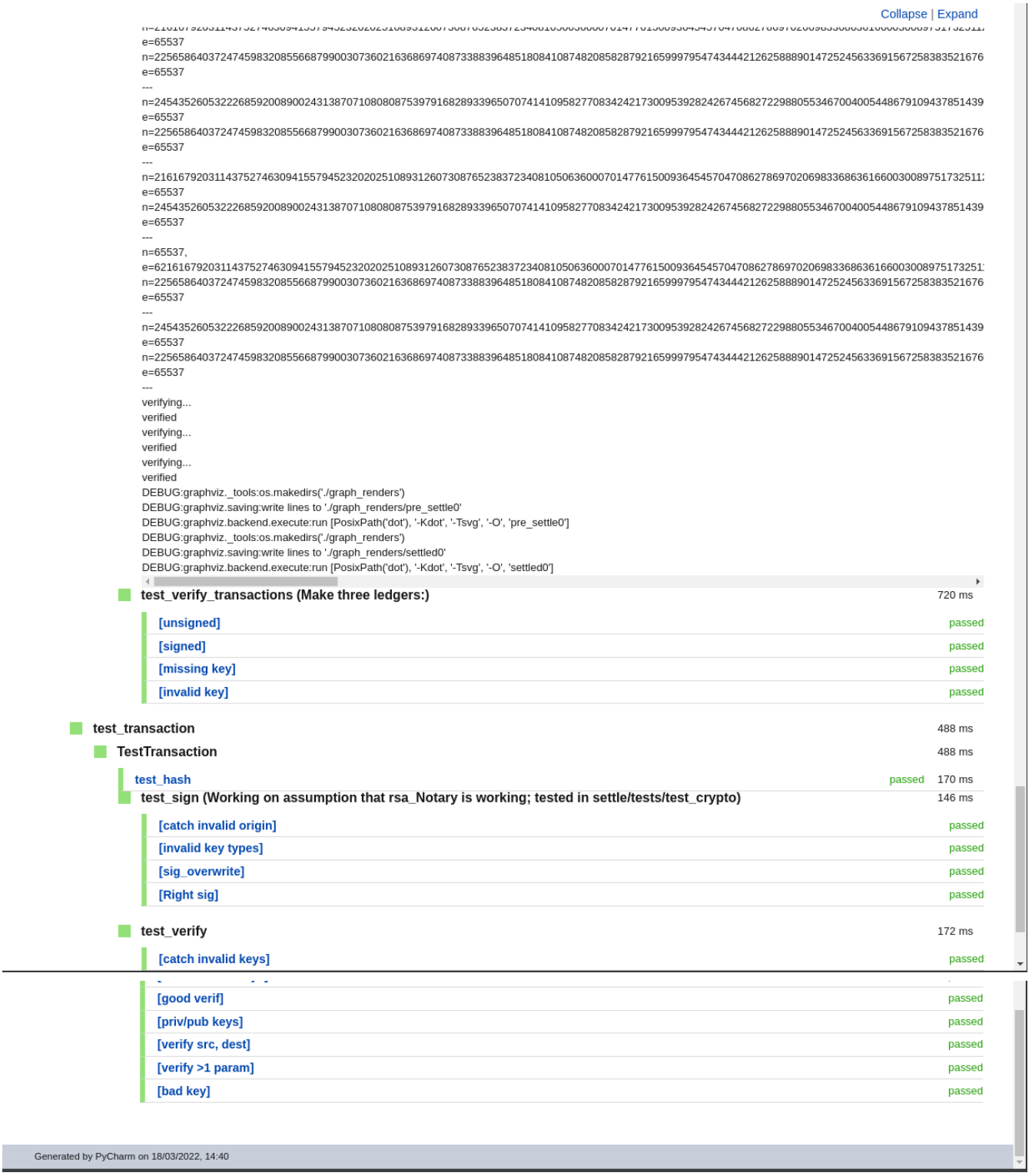
- test_ledger — 3.44 s
  - TestLedger — 3.44 s
    - test_add — 234 ms
      - [Add] — passed
      - [Catch non transaction] — passed
    - test_as_flow — 715 ms
      - [nodes] — passed
      - [to flow graph] — passed
    - test_flow_to_transactions — 551 ms
      - [to transactions] — passed
    - test_load_from_csv — passed — 285 ms

loading from csv
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514392
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216762
e=65537
---
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514392
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216762
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514392
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216762
e=65537
---
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514392
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216762

e=65537
---
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514390
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216760
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514390
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216760
e=65537
---
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514390
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216760
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514390
e=65537
---
n=65537,
e=6216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216760
e=65537
---
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514390
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216760
e=65537
---

test_simplify_ledger passed 937 ms

loading from csv
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514390
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216760
e=65537
---
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514390
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216760
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928242674568272298805534670040054486791094378514390
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112



Evidence of meeting Requirements - Section A & B

Crypto (A)		File	Test(s)
A1		test_crypto.test_hashes	*
A2		test_crypto.test_keys	
	A2.1		test_file_not_found test_file_loading
	A2.2		test_wrong_format
	A2.3		test_parsing test_unparsed_key test_unloaded_key test_public_key
A3			test_sign_verify
	A3.1		test_encryption
	A3.2		test_encryption
	A3.3		test_RSA_sign
	A3.4		test_RSA_sign
A4		test_transactions.test_transaction	
	A4.1		test_hash
	A4.2		test_sign
	A4.3		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	test_as_flow
B7			test_flow_to_transactions
B8			test_verify_transactions test_simplify_ledger

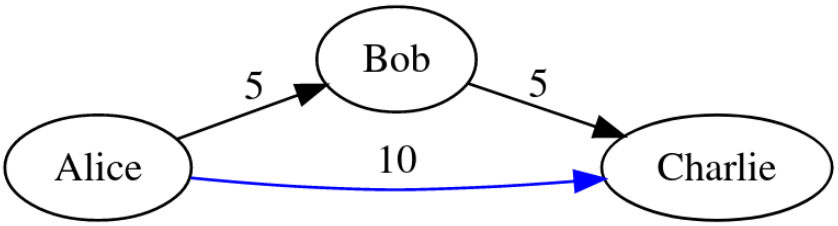
I appreciate this table may not show information about the individual tests. Every unit test I wrote has an informative docstring. Thus, refer to the code after the Evaluation for more clarification about the function of any unittest.

Proof of simplification algorithm functionality (B5)

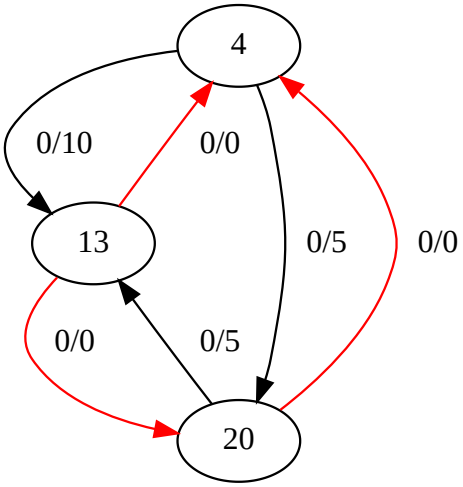
In the analysis section, I provided a hand-trace of the expected graph structures involved in settling a system of debts. Having done this hand tracing, I decided to use the same graph to test my algorithm.

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



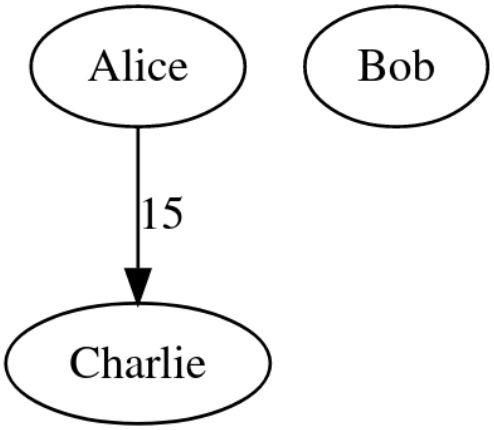
Generated



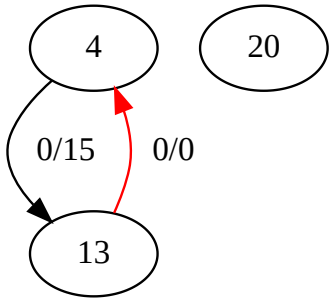
Since this is debug output, IDs are displayed instead of names and residual edges and flows are shown. Here, red edges can be ignored, and you can take the debt along the edge to equal the capacity of the edge.

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

Expected outcome



Generated outcome

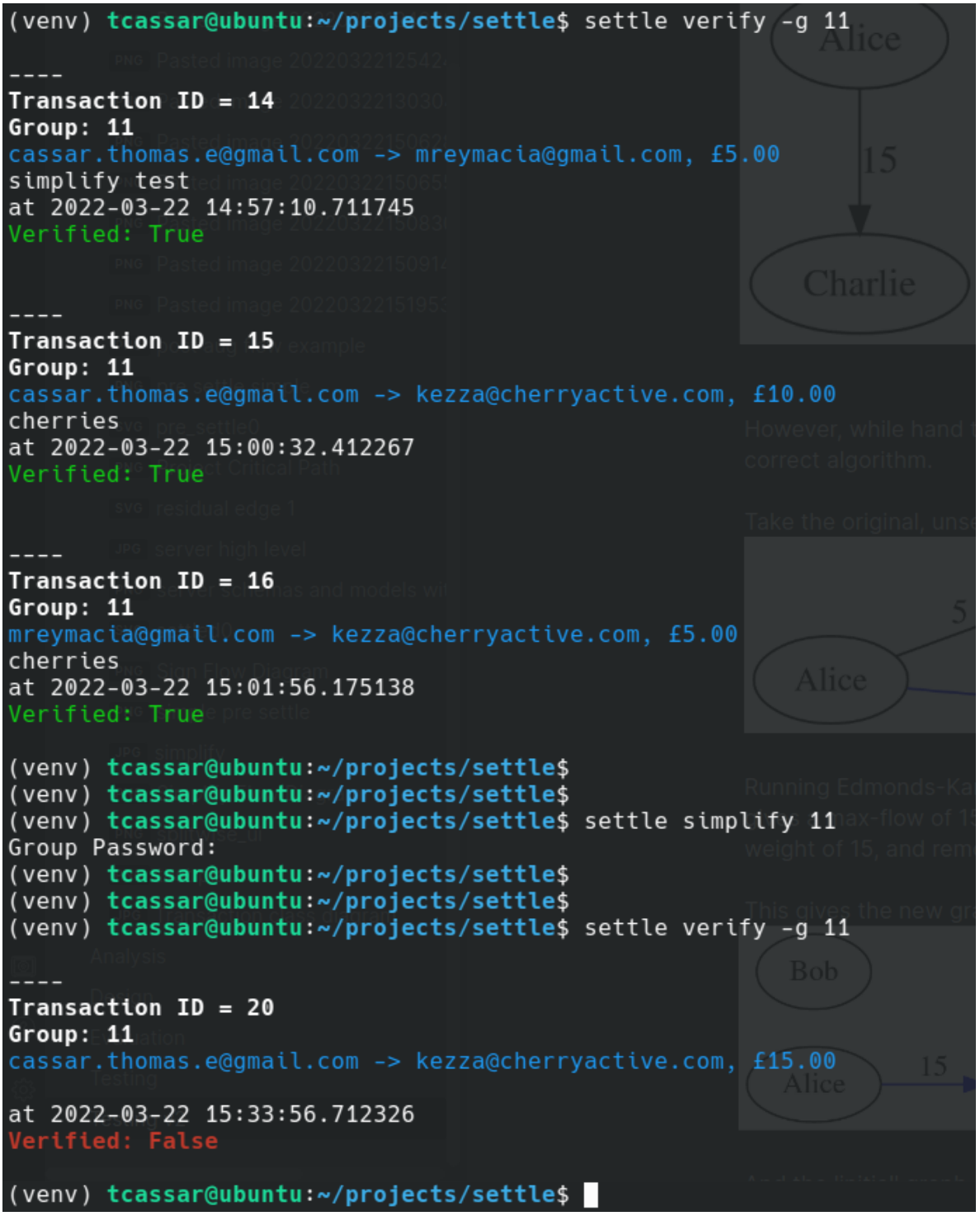


It is clear from these diagrams that the settling process has worked as expected, matching up perfectly with the hand-traced data. The same results are shown in the CLI of the technical solution

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



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!

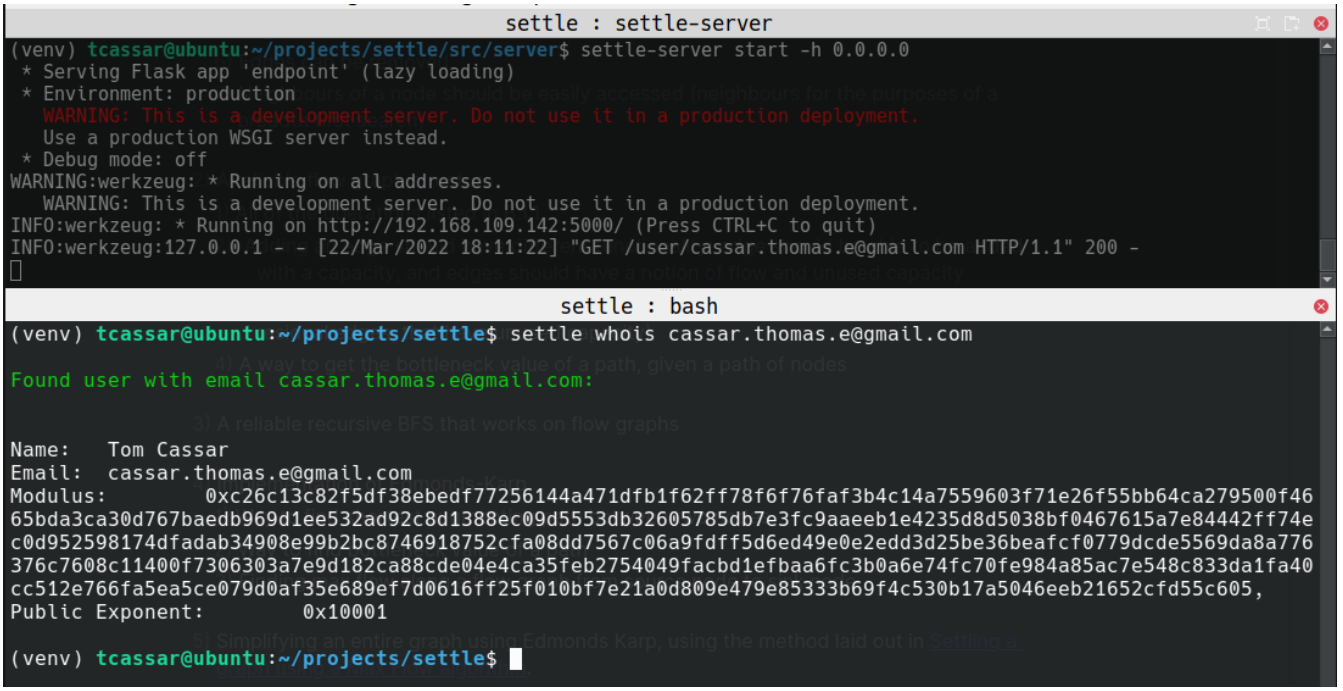
This example fulfils requirements **D1.6**, **D2.1**, **D2.4**, and **D2.5**

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

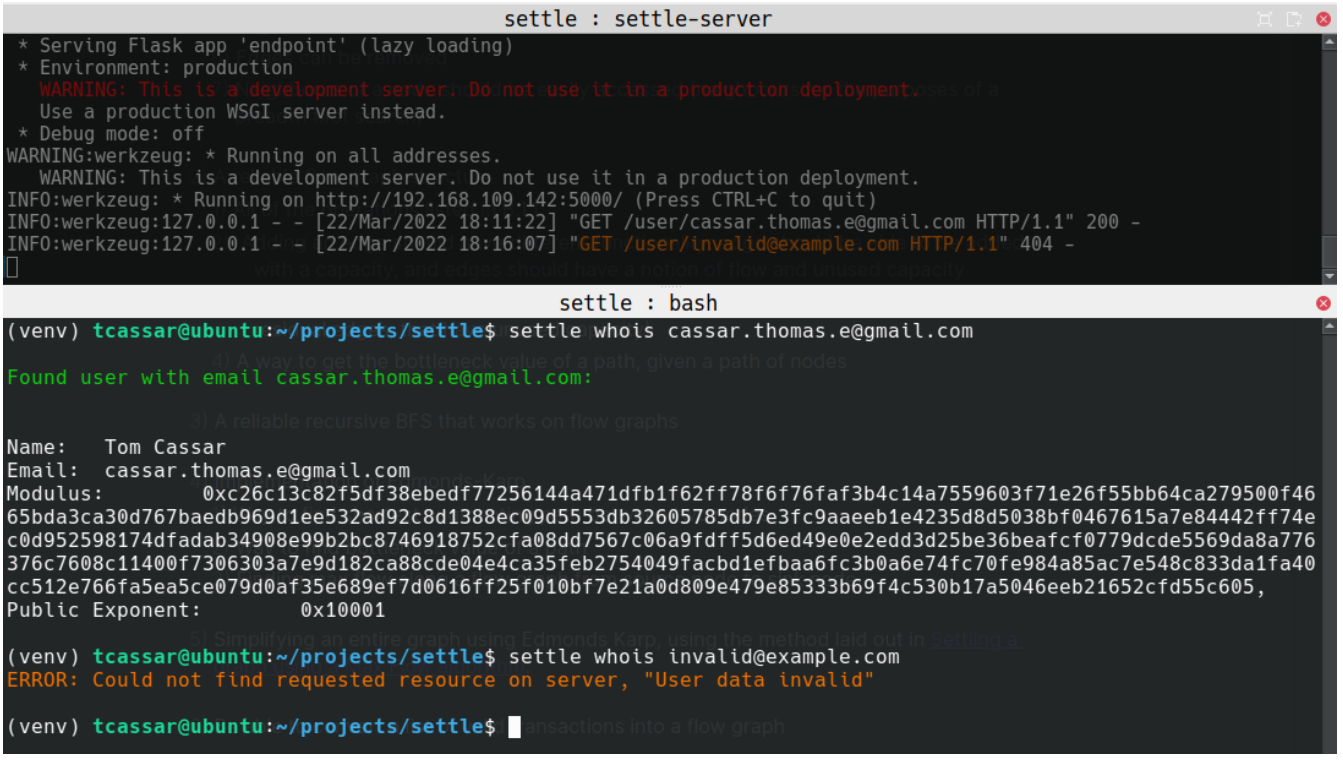
To show that my technical solution does in fact communicate over a REST API, I will show a simple get request made by the client and the corresponding logs produced by the server.



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.

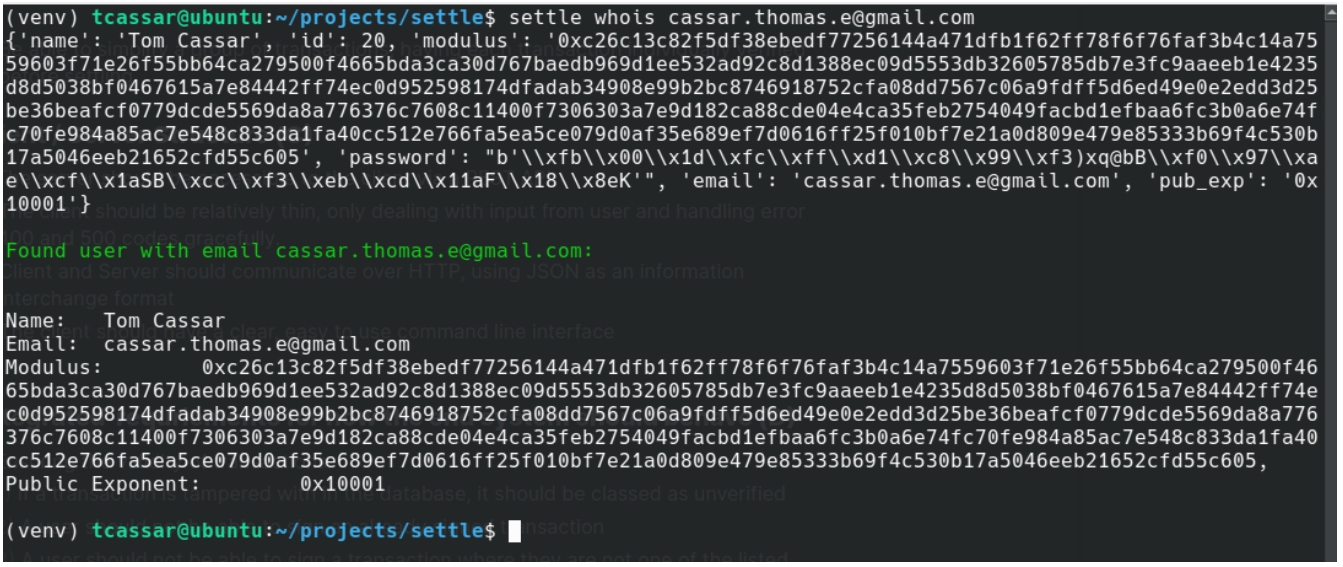


The server has responded with a 404 error code to the client's GET request. On the client side, this has been detected and raised. It has then been handled gracefully before showing the client the type of error (a resource not found), and what that might be (invalid user info).

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)



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:
        Key validated against server
Authorisation Error; aborting...
Failed to sign transaction: 13
Cannot overwrite signature of src
(venv) tcassar@ubuntu:~/projects/settle$
```

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

----
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

(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$
```

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

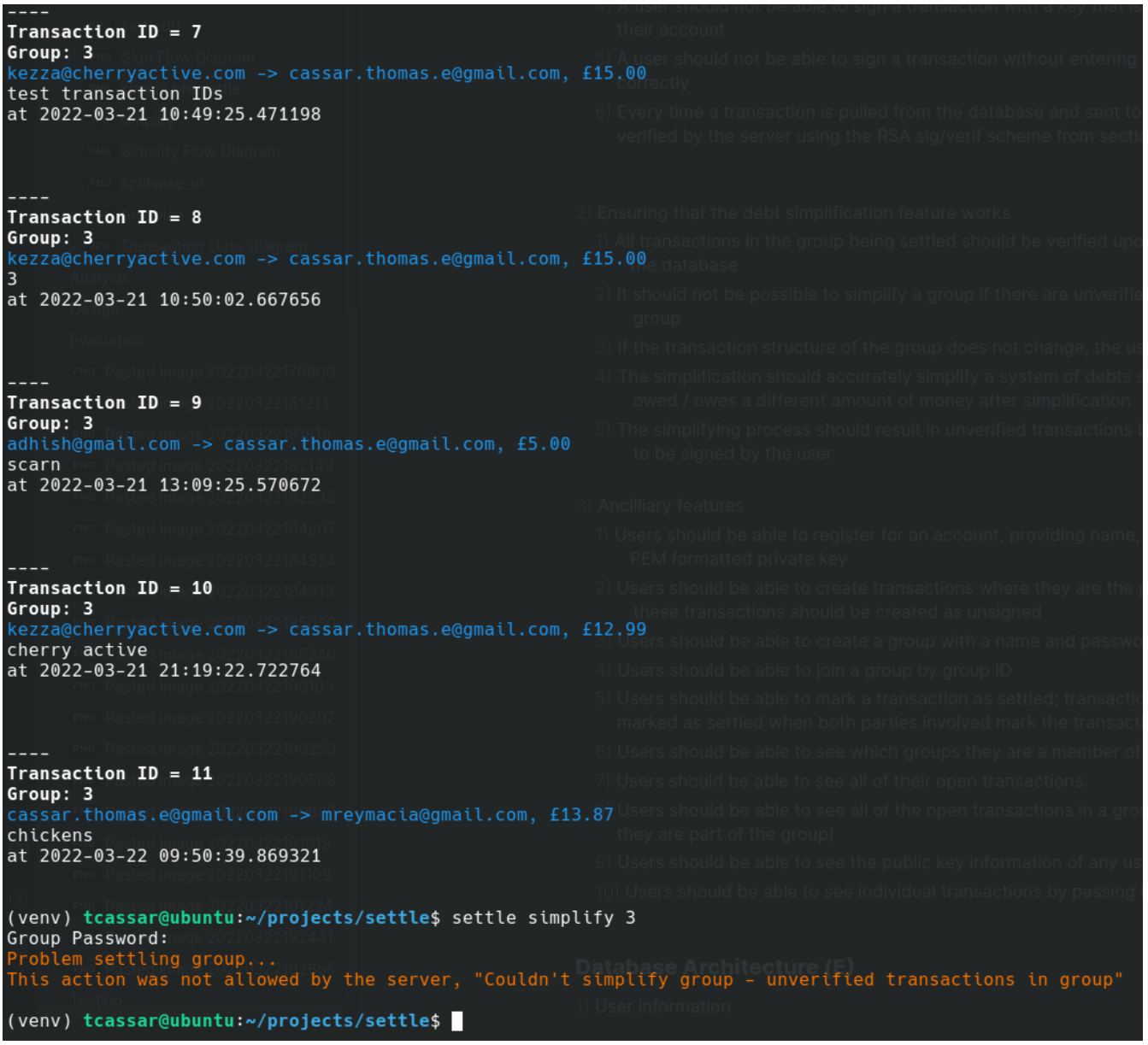
I attempt to sign the transaction with my private key, but I mistype my password **D1.5**

```
(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$
```

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



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

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

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.pe
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

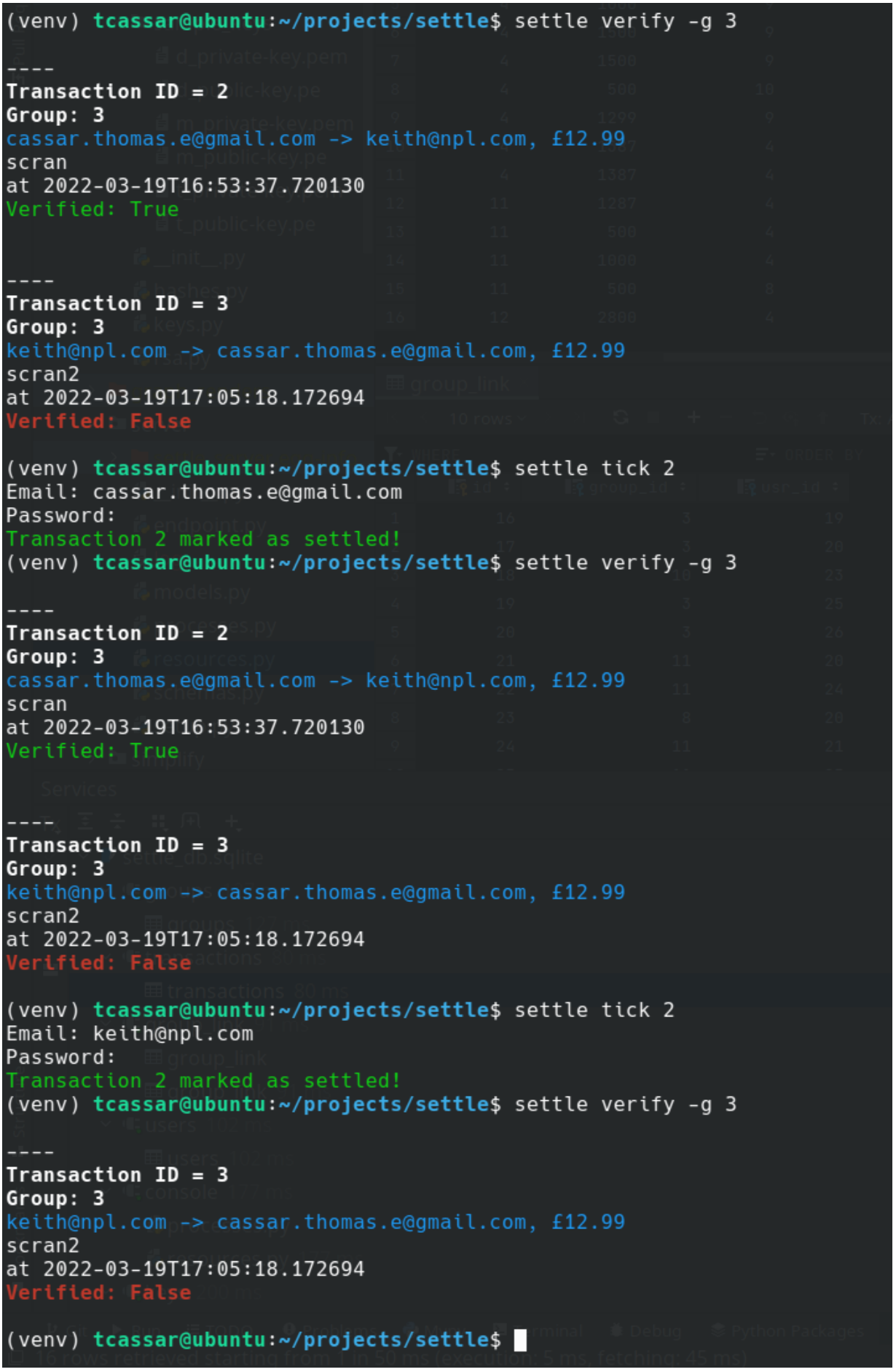
```
(venv) tcassar@ubuntu:~/projects/settle$ settle register
Full Name: example person
Email: example@gmail.com
Password:
Repeat for confirmation:
Path to RSA key: /home/tcassar/projects/settle/src/crypto/sample_keys/t_private-key.pem
Account created successfully

Name:   example person
Email:  example@gmail.com
Modulus:        0xc26c13c82f5df38ebedf77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30d767baed
b969d1ee532ad92c8d1388ec09d5553db32605785db7e3fc9aaeeb1e4235d8d5038bf0467615a7e84442ff74ec0d952598174dfadab34908e99b2bc874691
8752cfa08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beafcf0779dcde5569da8a776376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb275404
9facbd1efbaa6fc3b0a6e74fc70fe984a85ac7e548c833da1fa40cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f
4c530b17a5046eeb21652cfd55c605,
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:        0xc26c13c82f5df38ebedf77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30d767baed
b969d1ee532ad92c8d1388ec09d5553db32605785db7e3fc9aaeeb1e4235d8d5038bf0467615a7e84442ff74ec0d952598174dfadab34908e99b2bc874691
8752cfa08dd7567c06a9fdff5d6ed49e0e2edd3d25be36beafcf0779dcde5569da8a776376c7608c11400f7306303a7e9d182ca88cde04e4ca35feb275404
9facbd1efbaa6fc3b0a6e74fc70fe984a85ac7e548c833da1fa40cc512e766fa5ea5ce079d0af35e689ef7d0616ff25f010bf7e21a0d809e479e85333b69f
4c530b17a5046eeb21652cfd55c605,
Public Exponent:        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.



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**:

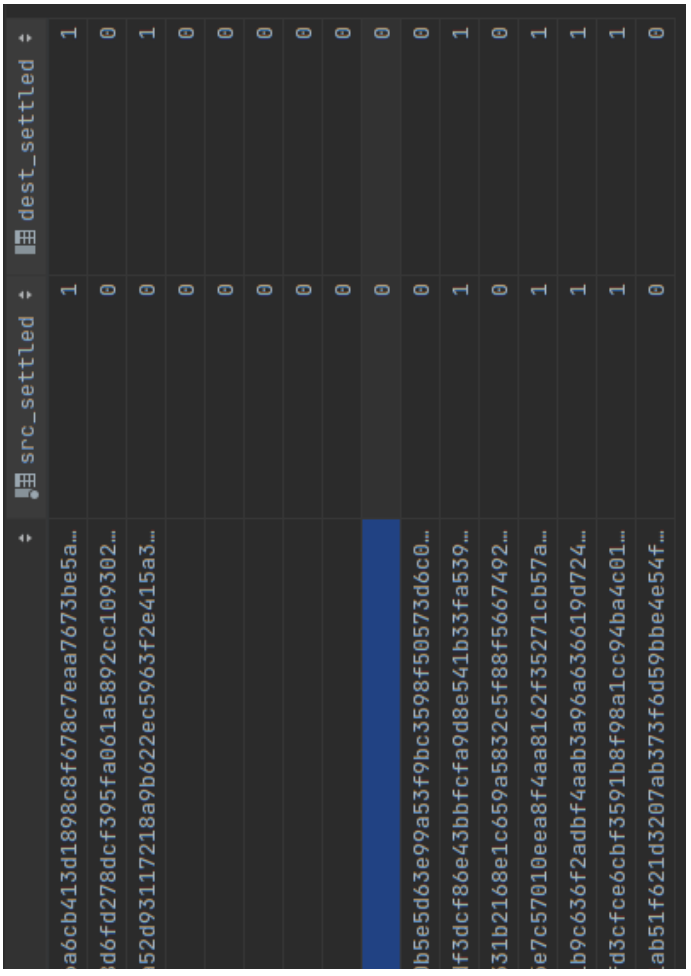
```
(venv) tcassar@ubuntu:~/projects/settle$ settle show -g
Email: cassar.thomas.e@gmail.com
Groups that you are a member of:

Group:
        Name: test group, ID = 3
Group:
        Name: simplify test, ID = 11
Group:
        Name: test, ID = 8
Group:
        Name: test d1.7, ID = 12

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

Evidence of Meeting Requirements - Section E

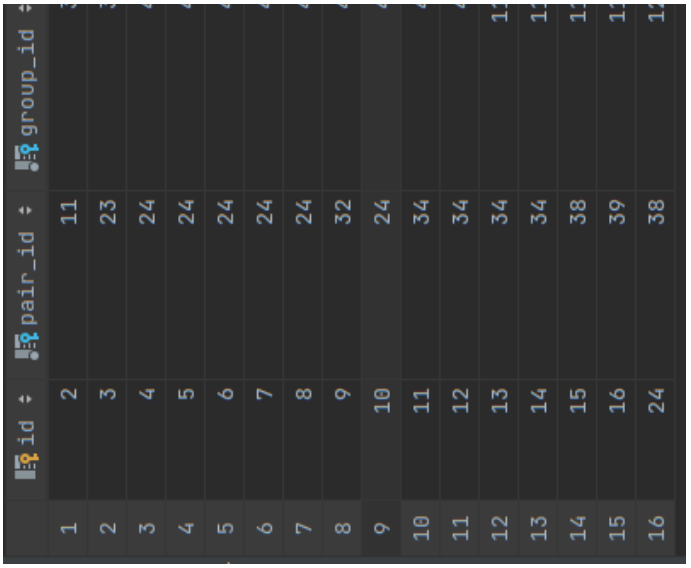
To show that I have implemented the database structure that I laid out in my requirements, I will screenshot the database tables, and briefly comment on which requirements each table fulfils.

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**



	amount	src_key	dest_key	reference	time_of_creation	src_sig	dest_sig
5	1299	4	5	scran	2022-03-19T16:53:37.720130	0x6619229faa043a839cc3595bf1284da28b58aa95b4e…	0xae6de6b18256
5	1299	5	4	scran2	2022-03-19T17:05:18.172694		0x55abc6445a38
4	1000	9	4	security transaction	2022-03-21 10:05:48.828933		0x9e0d44e0e8fa
4	1000	9	4	test	2022-03-21 10:44:16.839319		
4	1000	9	4	kez money	2022-03-21 10:47:25.463493		
4	1500	9	4	test transaction IDs	2022-03-21 10:49:25.471198		
4	1500	9	4	3	2022-03-21 10:50:02.667656		
4	500	10	4	scarn	2022-03-21 13:09:25.570672		
4	1299	9	4	cherry active	2022-03-21 21:19:22.722764		
4	1387	4	8	chickens	2022-03-22 09:50:39.869321		0x8f57244be550
4	1387	4	8	chickens	2022-03-22 09:55:42.800220	0x30a31c8375275d778542c9332f56e0232b83fc22750…	0xb1da9bcfe00d
1	1287	4	8	chickens to simplify	2022-03-22 12:41:46.332096	0x839075842746e0cb35b010118478dffccfc670311c6…	0x5f7fbabdf263
1	500	4	8	simplify test	2022-03-22 14:57:10.711745	0x249baaf2517307987f154619a99f136109357b285d6…	0x75d4f3e11805
1	1000	4	9	cherries	2022-03-22 15:00:32.412267	0x4bf53cfa1ab337b59dbf69e82905841f891a4c5dad4…	0x7f68499aed51
1	500	8	9	cherries	2022-03-22 15:01:56.175138	0x1fda9bf10034dad2689f4285b9af7de75adfa69616b…	0x77decc27c32f
2	2800	4	9	test one transactio…	2022-03-22 19:29:48.330112	0xaaa65d3ddd7dbd1baa1da6d74d6c87b778a59fe2354…	0x71c2db2484b1



Pairs

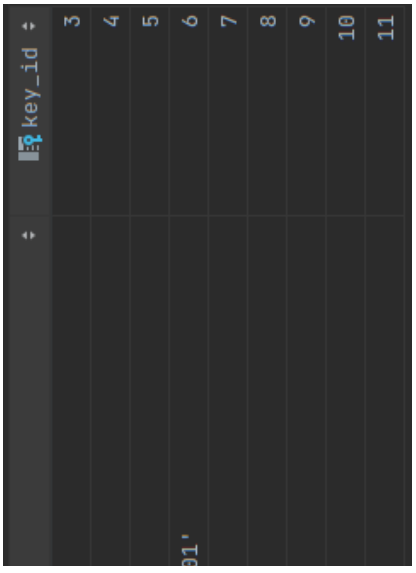
	id	src_id	dest_id
1	11	20	21
2	23	21	20
3	24	25	20
4	32	26	20
5	34	20	24
6	38	20	25
7	39	24	25

Keys

	id	n	e
1	3	0xc26c13c82f5df38ebedf77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30…	0x10001
2	4	0xc26c13c82f5df38ebedf77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30…	0x10001
3	5	0xb2c18e327280ff4abe3ed337b022518563a336871851178f9ee13cd8085c875c080184d836ca0a3f2d1fb771c98931cdbe2…	0x10001
4	6	0xb2c18e327280ff4abe3ed337b022518563a336871851178f9ee13cd8085c875c080184d836ca0a3f2d1fb771c98931cdbe2…	0x10001
5	7	0xc26c13c82f5df38ebedf77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30…	0x10001
6	8	0xb2c18e327280ff4abe3ed337b022518563a336871851178f9ee13cd8085c875c080184d836ca0a3f2d1fb771c98931cdbe2…	0x10001
7	9	0xab3cec18ae8df68c502ed71ce3376cca00c5e82aeebf9aa504794ed2b4b340093e381cbe2dc84c274da640a2d2c379a8c47…	0x10001
8	10	0xc26c13c82f5df38ebedf77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30…	0x10001
9	11	0xc26c13c82f5df38ebedf77256144a471dfb1f62ff78f6f76faf3b4c14a7559603f71e26f55bb64ca279500f4665bda3ca30…	0x10001

Users

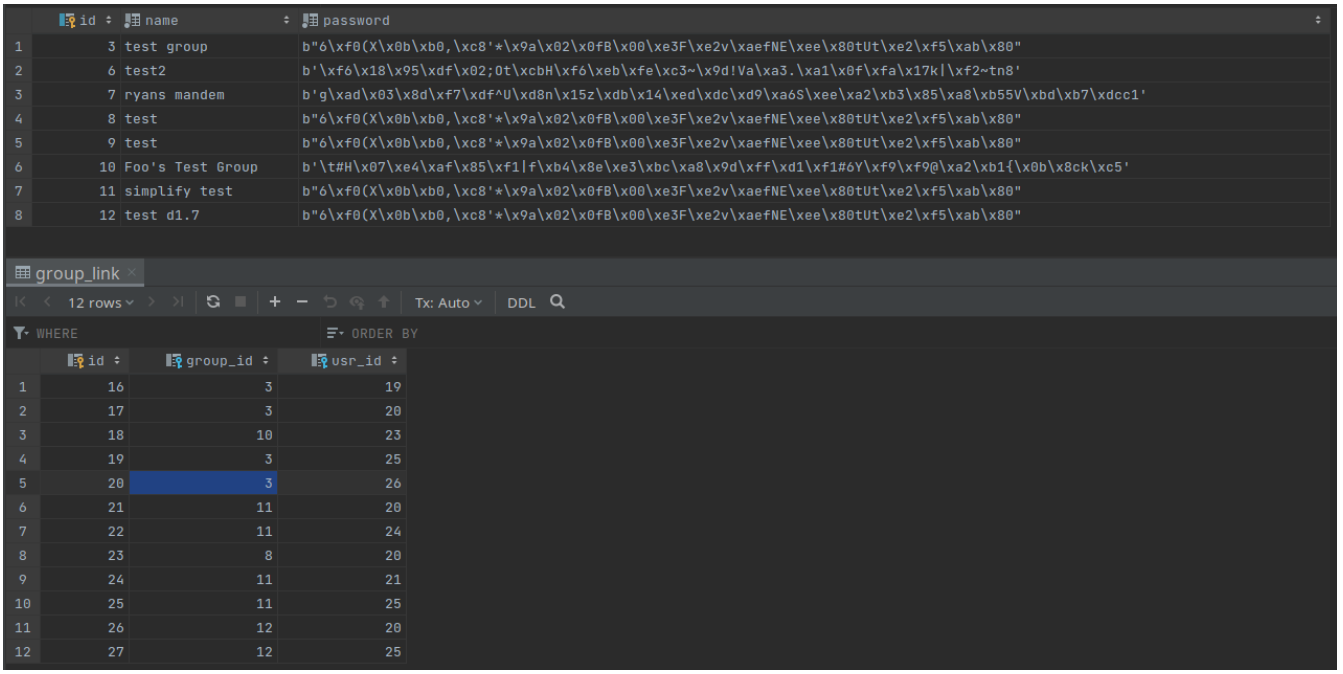
The users table, along with the keys table above, fulfil requirements **E1.1 → E1.5**



name	email	password
dmin	admin@admin.com	b'\xfb\x00\x1d\xfc\xff\xd1\xc8\x99\xf3)xq@bB\xf0\x97\xae\xcf\x1aSB\xcc\xf3\xeb\xcd\x11aF\x18\x8eK'
om Cassar	cassar.thomas.e@gmail.com	b'\xfb\x00\x1d\xfc\xff\xd1\xc8\x99\xf3)xq@bB\xf0\x97\xae\xcf\x1aSB\xcc\xf3\xeb\xcd\x11aF\x18\x8eK'
unes Tahir	keith@npl.com	b'\x90\xd9D\x8a6U\xc6<\x04\x92\x14Z\x86\xbd\xecl\xb2\x01K-H\xacTB\xdeD=M\xab\xd1\xf2\xf8'
oo	foo@bar.com	b'v\xd3\xbcA\xc9\xf5\x88\xf7\xfc\xd0\xd5\xbfG\x18\xf8\xf8K\x1cA\xb2\x08\x82p1\x00\xb9\xeb\x94\x13\x80|\x0
oo Bar	foobar@example.com	b'\t#H\x07\xe4\xaf\x85\xf1|f\xb4\x8e\xe3\xbc\xa8\x9d\xff\xd1\xf1#6Y\xf9\xf9@\xa2\xb1{\x0b\x8ck\xc5'
aia Rey Macia	mreymacia@gmail.com	b'8\xdd\x08\x8b5S\x81>\x8e0p/\xf6\x16o(\x97^\xc6E\xb0AN\xcdj\x89\n/0\x87^l'
ez Carey	kezza@cherryactive.com	b'\xff\x8es\xe7\xb3\x1f\x12\x1e\xeb\xbe46wZ\x81;\xf4`\xbf\xd1\xd8\xd0\xe4\x9dH]k\n\x14L\xcf~'
dhish	adhish@gmail.com	b'\xf5\x1a\xfe\x17\xc9\xe1\x85\xc9\xce\xb9\x9aoV\xd1/\x19K\xec\xe2m\x19c\xb9\tY\x86\x8eJ\x9c\xf4\x9bp'
xample person	example@gmail.com	b'p\x98=i/d\x81\x85\xfe\xbemo\xa6\x07c\n\xe6\x86I\xf7\xe6\xfcE\xb9F\x80\tl\x06\xe4\xfa\xdb'

Groups

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



	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'\xf6\x18\x95\xdf\x02;Ot\xcbH\xf6\xeb\xfe\xc3~\x9d!Va\xa3.\xa1\x0f\xfa\x17k|\xf2~tn8'
3	7	ryans mandem	b'g\xad\x03\x8d\xf7\xdf^U\xd8n\x15z\xdb\x14\xed\xdc\xd9\xa6S\xee\xa2\xb3\x85\xa8\xb55V\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'\t#H\x07\xe4\xaf\x85\xf1|f\xb4\x8e\xe3\xbc\xa8\x9d\xff\xd1\xf1#6Y\xf9\xf9@\xa2\xb1{\x0b\x8ck\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

12 rows · Tx: Auto · DDL

WHERE ORDER BY

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

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)

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.

```python
# coding=utf-8

"""
Unit tests for key handling
"""

import os
from unittest import TestCase

from src.crypto import keys


class TestRSAKeyLoading(TestCase):
    """Tests for RSA Keys"""

    def setUp(self) -> None:
        """Initialise loaders fresh between tests"""
        self.loader = keys.RSAKeyLoader()
        os.chdir("/home/tcassar/projects/settle/src")
        print(os.system("pwd"))
        self.key_path = "./crypto/sample_keys/d_private-key.pem"
        self.pub_key_path = "./crypto/sample_keys/d_public-key.pe"

    def test_file_loading(self):
        """Tests that file is being loaded correctly assuming correct file"""

        expected_start = "modulus"
        # test assumes if it starts off fine it will continue being fine
        received_start = self.loader.load(self.key_path)
        self.assertEqual(expected_start, received_start[:7])

    def test_parsing(self):
        """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
    test keys.
40          # lay out known values
41          k_e = 65537
42
43          k_d =
    42318605026103475553526985044230475639442568138229963925734038989576569874864579669508371562955146126547877955291914430323301128393346172685767442683818738967216638476720764074944647817540070989695696588210951082558372250397158894945714319204774976709066583095965638944919595239862657801000401988151966317962866310493446444221800190039054548479140813630569444720847689992964879750244948034517871603890465032993904165597478235001604038074703682680413388703757501362444680122222701317562224831854480644662034354542275281227314387046825525570376824278684460211954070416261823570911887469407283866423584255451231566190543
44          k_n =
    21616792031143752746309415579452320202510893126073087652383723408105063600070147761500936454570470862786970206983368636166003008975173251124763374054321346030354457021425165356037781636930036106404418295413726431002556836900067049867944499043128421557451025437279819137427553642625019821057158620227234339857381390676943784764665144556770107968670906167894855774586373708692617743377046549318417755362244202417503903181827421441408785602795322818434551136750209831842525036691384924518828835962027759009110005026961294337702970618450353224231935687225848095890069620605874219546032017844978461582635821441774306426043
45
46          # build lists for testing
47          cases = [
48              "pub_exp",
49              "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,
    known, received):
74              with self.subTest(test):
75                  self.assertEqual(rec_val, known_val)
76                  self.assertEqual(type(rec_val), type
    (known_val))
77
78      def test_file_not_found(self):
79          path = "./adsads"
80
81          with self.assertRaises(keys.RSAParserError):
82              loader = self.loader
83              loader.load(path)
84
85      def test_wrong_format(self):
86          """Checks that we can deal with files being
    the wrong format"""
87          with self.assertRaises(keys.RSAParserError):
```

```
 88             self.loader.load("./crypto/sample_keys/
    d_public-key.pe")
 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
```

```python
# coding=utf-8
import hashlib
import sys
from unittest import TestCase

from src.crypto import hashes


class TestHash(TestCase):
    """
    Test hash interfaces (functionality not aim of testing as functionality comes from hashlib
    1) Check initialisation as expected
    2) Check update
    3) Check digest
    4) Check hexdigest == intdigest
    """

    def test_init(self) -> None:
        """Checks that _hasher is initialised correctly i.e. no strange start values"""
        h = hashes.Hasher()
        self.assertEqual(
            h.digest().int_digest(),
            int.from_bytes(hashlib.sha3_256(b"").digest(), byteorder=sys.byteorder),
        )  # should be initialised empty

    def test_hash(self) -> None:
        """tests that hash object can validate hash looking numbers"""
        with self.subTest("valid"):
            h_val = hashlib.sha3_256(b"1234").digest()
            h = hashes.Hash(h_val)  # create a definitely valid hash

        with self.subTest("too short"):
            with self.assertRaises(hashes.HashError):
                hashes.Hash(b"12")
```

```
35
36          with self.subTest("too long"):
37              with self.assertRaises(hashes.HashError
    ):
38                  hashes.Hash(
39                      b"
    1157920892373161954235709850086879078532699846656405
    6403945758400791312963993 60"
40                  )
41
42          with self.subTest("wrong type"):
43              with self.assertRaises(hashes.HashError
    ):
44                  # noinspection PyTypeChecker
45                  hashes.Hash("dave")
46
47      def test_update_digest(self) -> None:
48          """Ensures that a hash with a given value
    will digest the correct thing"""
49          h = hashes.Hasher(b"1234")
50          h_ = hashlib.sha3_256(b"1234")
51
52          with self.subTest("before update"):
53              self.assertEqual(h.digest().h, h_.digest
    ())
54
55          update_msg = b"test hash update"
56          h.update(update_msg)
57          h_.update(update_msg)
58
59          with self.subTest("after update"):
60              self.assertEqual(h.digest().h, h_.digest
    ())
61
62      def test_hasher_fails(self):
63          """Checks that hasher objects when not bytes
     are passed into it"""
64          with self.assertRaises(hashes.HasherError):
65              # noinspection PyTypeChecker
66              hasher = hashes.Hasher("abd")
67
```

```python
# coding=utf-8

"""
Testing sign / verify through RSA working as expected
"""
import os
from unittest import TestCase

from src.crypto import keys
from src.crypto import rsa
from src.transactions.transaction import Signable


def setUpModule():
    os.chdir("/home/tcassar/projects/settle/src")


class TestRSA(TestCase):
    """Just tests RSA parts"""

    def setUp(self) -> None:
        # load keys
        ldr = keys.RSAKeyLoader()
        ldr.load("./crypto/sample_keys/d_private-key.pem")
        ldr.parse()

        # build public and private
        self.private = keys.RSAPrivateKey(ldr)
        self.public = keys.RSAPublicKey(ldr)

    def test_encryption(self):
        """Checks to see if process is reversible, and encrypted is different to how it started"""
        message = " | maia"
        m_bytes = bytes(message, encoding="utf8")
        encrypted = rsa.RSA.encrypt(m_bytes, self.public)
        # TODO: actually fix byte overflow affair
        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.
   DecryptionError):
41                 _ = rsa.RSA.naive_decrypt(encrypted
   , self.public)  # type: ignore
42
43         with self.subTest("encrypted"):
44             self.assertNotEqual(m_bytes, encrypted)
45
46         with self.subTest("Successful decryption"):
47             self.assertEqual(message, decrypted)
48
49     def test_RSA_sign(self):
50         """Checks for consistent creating /
   verifying of a 'signature'"""
51
52         message = " | maia"
53         m_bytes = message.encode("utf8")
54         sig: bytes = rsa.RSA.sign(m_bytes, self.
   private)
55         de_sign: bytes = rsa.RSA.inv_sig(sig, self.
   public)
56
57         self.assertEqual(m_bytes, de_sign)
58
```

```python
# coding=utf-8
from unittest import TestCase

from src.simplify.base_graph import Digraph
from src.simplify.flow_graph import FlowGraph
from src.simplify.graph_objects import Vertex
from src.simplify.path import Path, prev_map, disc_map, BFSQueue
from src.simplify.weighted_digraph import WeightedDigraph


class TestPath(TestCase):
    """Tests searching for BFS"""

    def setUp(self):
        # make a graph with 5 nodes

        labels = ["a", "b", "c", "d", "e", "f"]
        self.vertices = [Vertex(ID, label=label) for ID, label in enumerate(labels)]

        self.g = Digraph(self.vertices)
        a, b, c, d, e, f = self.vertices
        self.g.add_edge(a, b, c)
        self.g.add_edge(b, d)
        self.g.add_edge(d, f)
        self.g.add_edge(c, e)
        self.g.add_edge(d, f)
        self.g.add_edge(e, f, b)

        # set up weighted_graph and flow graphs
        self.weighted_graph = WeightedDigraph(self.vertices)
        self.flow_graph = FlowGraph(self.vertices)

        for g in [self.weighted_graph, self.flow_graph]:
            g.add_edge(a, (b, 10), (c, 10))
            g.add_edge(b, (d, 25))
            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
    path along
43          """
44
45          a, b, c, d, e, f = self.vertices
46
47          cases = ["digraph", "weighted_graph", "flow"
    ]
48
49          # check precalulated shortest path
50          graph_cases = [self.g, self.weighted_graph,
    self.flow_graph]
51
52          for case, g in zip(cases, graph_cases):
53              with self.subTest(case):
54                  calc_shorted: list[Vertex] = Path.
    shortest_path(
55                      self.g, a, f, self.g.neighbours
56                  )
57                  expected: list[Vertex] = [a, c, e, f
    ]
58                  self.assertEqual(expected,
    calc_shorted)
59
60      def test_build_path(self):
61          """Given a prev_map, check we build the
    right path"""
62
63          # generate vertices, unpack into vars
64          vertices = []
65          for ID, name in enumerate(["A", "B", "C", "D
    ", "E", "F"]):
66              vertices.append(Vertex(ID, name))
67
68          a, c, b, d, e, f = vertices
69
```

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

```
103         """Checks that BFS can stop when its given
    target is found"""
104         graph = self.flow_graph
105         a, b, c, *_ = graph.nodes()
106 
107         expected = {node: None for node in graph.
    nodes()}
108         expected[b] = a
109         expected[c] = a
110 
111         queue, discovered, previous = Path.
    build_bfs_structs(graph, a)
112 
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
    for BFS are built correctly"""
125         with self.subTest("with initial value"):
126             queue, disc, prev = Path.
    build_bfs_structs(
127                 self.flow_graph, self.vertices[0]
128             )
129             self.assertEqual(queue, BFSQueue(self.
    vertices[0]))
130 
131         with self.subTest("with initial value"):
132             queue, disc, prev = Path.
    build_bfs_structs(self.flow_graph)
133             self.assertEqual(queue, BFSQueue())
134 
```

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

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

```
33          with self.subTest("exceed capacity"), self.
   assertRaises(FlowEdgeError):
34              self.edges[1].push_flow(3)
35
36      def test_adjust_edge(self):
37          """Tests that the adjust_edges function
   works as expected
38          i.e when adjust is called, edges in the
   graph morph to edges with a flow equal to their
   unused capacity
39          and all edges of weight 0 are deleted"""
40          (
41              res,
42              fwd,
43          ) = self.edges
44          fwd.push_flow(3)
45          res.push_flow(-3)
46
47          new_fwd = FlowEdge(Vertex(0), Vertex(1), 2)
48          new_res = FlowEdge(Vertex(0), Vertex(1), 0)
49
50          fwd.adjust_edge()
51          res.adjust_edge()
52
53          self.assertEqual(new_fwd, fwd)
54          self.assertEqual(new_res, res)
55
56          # catch EdgeCapacityZero
57          with self.assertRaises(EdgeCapacityZero):
58              fwd.push_flow(2)
59              fwd.adjust_edge()
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))
   for n in range(5)]
66          self.graph = FlowGraph(self.nodes)
67
68      def test_is_edge(self):
```

```
 69          a, b, c, d, e = self.nodes
 70
 71          with self.subTest("no edge"):
 72              self.assertFalse(self.graph.is_edge(a,
    b))
 73
 74          with self.subTest("edge"):
 75              self.graph.add_edge(a, (b, 4))
 76              self.assertTrue(self.graph.is_edge(a, b
    ))
 77
 78      def test_get_edge(self):
 79          a, b, c, d, e = self.graph.nodes()
 80          self.graph.add_edge(a, (b, 5))
 81          self.assertEqual(FlowEdge(a, b, 5), self.
    graph.get_edge(a, b))
 82
 83      def test_add_edge(self):
 84          a, b, c, d, e = self.nodes
 85          # check there aren't edges
 86          with self.subTest("negative test"):
 87              self.assertFalse(self.graph.is_edge(a,
    b))
 88              self.assertFalse(self.graph.is_edge(b,
    a, residual=True))
 89          self.graph.to_dot()
 90          self.graph.add_edge(a, (b, 5))
 91          self.graph.to_dot()
 92          with self.subTest("added"):
 93              self.assertTrue(self.graph.is_edge(a, b
    ))
 94              self.assertTrue(self.graph.is_edge(b, a
    , residual=True))
 95              self.assertFalse(self.graph.is_edge(b,
    a))
 96
 97          with self.subTest("Net debt (adding)"):
 98              self.assertEqual(
 99                  {
100                      Vertex(ID=0, label="a"): 5,
101                      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
    capacity, including residual
140         These edges count as valid paths to
    neighbouring nodes in Edmonds-Karp"""
141         graph = self.graph
142         a, b, c, d, *_ = graph.nodes()
143         graph.add_edge(a, (b, 10))
144         graph.add_edge(b, (c, 2))
145         graph.add_edge(c, (d, 10))
146         graph.add_edge(d, (a, 10))
147 
148         graph.to_dot()
149 
150         MaxFlow.augment_flow(graph, [a, b, c, d], 2
    )
151         c_flow_neighbours = graph.flow_neighbours(c
    )
152         self.assertEqual([d, b], GenericDigraph.
    nodes_from_edges(c_flow_neighbours))
153 
154     def test_bool(self):
155         """Tests that dunder bool method returns
    true when there are edges in the graph"""
156         # empty
157         print(self.graph.graph)
158         self.assertFalse(bool(self.graph))
159 
160         # with an edge
161         a, b, c, d, e = self.nodes
162         self.graph.add_edge(a, (b, 10))
163 
164     def test_add_existing(self):
165         """Tests that adding an existing edge will
    add capacity onto the existing edge,
166         as opposed to adding a new distinct edge
    from src -> dest"""
167         a, b, c, d, e = self.graph.nodes()
168         self.graph.add_edge(a, (b, 5))
169         self.graph.add_edge(a, (b, 5))
170         print(self.graph)
```

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

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

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

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

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

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

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

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

```python
# coding=utf-8

from unittest import TestCase

from src.simplify.base_graph import Digraph
from src.simplify.flow_graph import *
from src.simplify.graph_objects import Edge
from src.simplify.weighted_digraph import
WeightedDigraph


class TestDigraph(TestCase):
    """Generate a digraph and some vertices"""

    def setUp(self) -> None:
        """Build basic graph"""
        labels = ["u", "v", "w"]
        self.vertices = [Vertex(ID, label=label) for
 ID, label in enumerate(labels)]

        self.graph = Digraph(self.vertices)
        u, v, w = self.vertices
        self.graph.add_edge(u, v, w)
        self.graph.add_edge(v, w)

    def test_init(self):
        expected = "U -> VW\nV -> W\nW -> \n"

        with self.subTest("init"):
            self.assertEqual(expected, str(self.
graph))

        with self.subTest("helpers"):
            self.assertTrue(self.graph.is_node(self.
vertices[0]))
            self.assertTrue(self.graph.sanitize(*
self.vertices))

            with self.assertRaises(GraphError):
                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
    [v])
62
63      def test_pop_edge(self):
64          u, v, w = self.vertices
65          self.assertTrue(self.graph.is_edge(u, v))
66          self.graph.pop_edge(u, v)
67
68          self.assertFalse(self.graph.is_edge(u, v))
69
70      def test_add_edge(self):
71          u, v, w = self.graph.nodes()
72          self.assertFalse(self.graph.is_edge(w, v))
73          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)
    for ID, label in enumerate(labels)]
 87
 88         self.graph = WeightedDigraph(self.vertices)
 89         u, v, w = self.vertices
 90         self.graph.add_edge(u, (v, 1), (w, 2))
 91         self.graph.add_edge(v, (w, 3))
 92
 93     def test_add_edge(self):
 94         u, v, w = self.vertices
 95         self.assertFalse(self.graph.is_edge(w, v))
 96         self.graph.add_edge(w, (v, 4))
 97         self.assertTrue(self.graph.is_edge(w, v))
 98
 99     def test_add_existing_edge(self):
100         u, v, w = self.graph.nodes()
101         self.assertEqual(self.graph.is_edge(v, w),
    3)
102         self.graph.add_edge(v, (w, 2))
103         self.assertEqual(self.graph.is_edge(v, w),
    5)
104
105     def test_flow_through(self):
106         for node, flow in zip(self.vertices, [3, 2
    , -5]):
107             with self.subTest(node):
108                 self.assertEqual(self.graph.
    flow_through(node), flow)
```

```
1 ID,src,dest,amount,group,src_n,src_e,dest_n,dest_e,
  src_d,dest_d
2 0,04,20,5,0,
  216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511247633740543213460303544570214251653560377816369300361064044182954137264310025568369000670498679444999043128421557451025437279819137427553642625019821057158620227234339857381390676943784764665144556770107968670906167894855774586373708692617743377046549318417755362244202417503903181827421441408785602795322818434551136750209831842525036691384924518828835962027759009110005026961294337702970618450353224231935687225848095890069620605874219546032017844978461582635821441774306426034,65537,
  24543526053222685920089002431387071080808753979168289339650707414109582770834242173009539282426745682722988055346700400544867910943785143909523891526583685873084778519448472887002660212182814126070863004440351914152540555901097499930235793743416545610368040032596055789846809140241500787296582367699342532180447446042851559187629052238168999819843726156363830030069139123051200441111945407276884741033640730462395861985143994176991609920694746571704707094532698910844357727419783667461542340930287506515793569704149384784131969168222943901917462927710491717935881329934336642631167278369058626398091411958822782486021,65537,
  42318605026103475553526985044230475639442568138229963925734038989576569874864579669508371562955146126547877955291914430323301128393346172685767442683818738967216638476720764074944647817540070989695696588210951082558372250397158894945714319204774976709066583095965638944919595239862657801000401988151966317962866310493446444221800190039054548479140813630569444720847689992964879750244948034517871603890465032993904165597478235001604038074703682680413388703757501362444680122222701317562224831854480644662034354542275281227314387046825525570376824278684460211954070416261823570911887469407283866423584255451231566190517451642798421916838978706887752529294378563794943959644593481018317996812806135240585387347011403463919
```

```
2
484312360288671519765089185961940677538082993405248358724852816062664396208614460346356077862920426734827642392974806673957359914994399933521690425914379137265608737153994017658024271602617427327911897090105542806007602652290786721234528626131896832489562551558690301697229495263298457688434655931885453437842161128989953202272867367837891847562410504381902662947168208004596529501772278874463251649170353674520402106924748077589683971300268994119580437205561936946700600893560778543423518093933835790069197852673
3 1,04,13,10,0,
216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511247633740543213460303544570214251653560377816369300361064044182954137264310025568369000670498679444999043128421557451025437279819137427553642625019821057158620227234339857381390676943784764665144556770107968670906167894855774586373708692617743377046549318417755362244202417503903181827421441408785602795322818434551136750209831842525036691384924518828835962027759009110005026961294337702970618450353224231935687225848095890069620605874219546032017844978461582635821441774306426 03,65537,
225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676090489827618604882344659937010912157571801294412422946171359122243583308263282848559863033353276205119720064531420906643446610551234084941258572108217677529623697862636846419040961276021458997843951217963241478789474394699527160580771635153728361835033613801478959059921454822591898567781991786261932874106056739293869626830428217313621069552445747883424421554798845480580652219017109280678813179596281553889038339230703284433283787263864958666155092517689554259271552956245888 23,65537,
42318605026103475553526985044230475639442568138229963925734038989576569874864579669508371562955146126547877955291914430323301128393346172685767442683818738967216638476720764074944647817540070989695696588210951082558372250397158894945714319204774976709066583095
```

```
3
   965638944919595239862657801000401988151966317962866310
   493446644422180019003905454847914081363056944472084768
   999296487975024494803451787160389046503299390416559747
   823500160403807470368268041338870375750136244468012222
   701317562224831854480644662034354542275281227314387046
   825525570376824278684460211954070416261823570911887469
   407283866423584255451231566190577541428218992751486727
   644862498271472919514635342583131191306090515724089556
   837205820094594019006617381787091438615311514626770508
   215330966596440453041517774718735698700608641179259809
   996868147443542318468896990846846049836207197701758385
   280007526246224538969096382406756380016525685903567419
   429111431200416762356947622267619084456004720872031090
   989474470600876826405729239073983735694492507804544435
   298169779944004811276378982537638262601152485343855539
   104875876907807892323552073225968951530325990045227310
   213319704081395654861368829855439175850787537877943284
   782407352583525021787084903098686611464144261278387324
   2,20,13,5,0,24543526053222685920089002431387071080808
   753979168289339650707414109582770834242173009539282426
   745682722988055346700400544867910943785143909523891526
   583685873084778519448472887002660212182814126070863004
   440351914152540555901097499930235793743416545610368040
   032596055789846809140241500787296582367699342532180447
   446042851559187629052238168999819843726156363830030069
   139123051200441111945407276884741033640730462395861985
   143994176991609920694746571704707094532698910844357727
   419783667461542340930287506515793569704149384784131969
   168222943901917462927710491717935881329934336642631167
   278369058626398091411958822782486021,65537,225658640372
   474598320855668799003073602163686974087338839648518084
   108748208582879216599979547434442126258889014725245633
   691567258383521676090489827618604882344659937010912157
   571801294412422946171359122243583308263282848559863033
   353276205119720064531420906643446610551234084941258572
   108217677529623697862636846419040961276021458997843951
   217963241478789474394699527160580771635153728361835033
   613801478959059921454822599
```

```
4
  189856778199178626193287410605673929386962683042821731362106955244574788342442155479884548058065221901710928067881317959628155388903833923070328443328378726386495866615509251768955425927155295624588823,65537,1745164279842191683897870688775252929437856379494395964459348101831799681280613524058538734701140346391948431236028867151976508918596194067753808299340524835872485281606266439620861446034635607786292042673482764239297480667395735991499439993352169042591437913726560873715399401765802427160261742732791189709010554280600760265229078672123452862613189683248956255155869030169722949526329845768843465593188545343784216112898995320227286736783789184756241050438190266294716820800459652950177227887446325164917035367452040210692474807758968397130026899411958043720556193694670060089356077854342351809393383579006919785267360077543428218992751486727644862498271472919514635342583131191306090515724089556837205820094594019006617381787091438615311514626770508215330966596440453041517774718735698700608641179259809996868147443542318468896990846846049836207197701758385280007526246224538969096382406756380016525685903567419429111431200416762356947622267619084456004720872031090989474470600876826405729239073983735694492507804544435298169779944004811276378982537638262601152485343855539104875876907807892323552073225968951530325990045227310213319704081395654861368829855439175850787537877943284782407352583525021787084903098686611464144261278387358352502178708490309868661146414426127838735 3,04,20,5,1,2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112476337405432134603035445702142516535603778163693003610640441829541372643100255683690006704986794449990431284215574510254372798191374275536426250198210571586202272343398573813906769437847646651445567701079686709061678948557745863737086926177433770465493184177553622442024175039031818274214414087856027953228184345511367502098318425250366913849245188288359620277590091100050269612943377029706184503532242319356872258480958900696206058 7
```

```
5 42195460320178449784615826358214417743064260 3,65537,
  24543526053222685920089002431387071080808753979168289
  33396507074141095827708342421730095392824267456827229
  88055346700400544867910943785143909523891526583685873
  08477851944847288700266021218281412607086300444035191
  41525405559010974999302357937434165456103680400325960
  55789846809140241500787296582367699342532180447446042
  85155918762905223816899981984372615636383003006913912
  30512004411119454072768847410336407304623958619851439
  94176991609920694746571704707094532698910844357727419
  78366746154234093028750651579356970414938478413196916
  82229439019174629277104917179358813299343366426311672
  78369058626398091411958822782486021,65537,42318605026
  10347555352698504423047563944256813822996392573403898
  95765698748645796695083715629551461265478779552919144
  30323301128393346172685767442683818738967216638476720
  76407494464781754007098969569658821095108255837225039
  71588949457143192047749767090665830959656389449195952
  39862657801000401988151966317962866310493446444221800
  19003905454847914081363056944472084768999296487975024
  49480345178716038904650329939041655974782350016040380
  74703682680413388703757501362444680122222701317562224
  83185448064466203435454227528122731438704682552557037
  68242786844602119540704162618235709118874694072838664
  23584255451231566190 5,
  17451642798421916838978706887752529294378563794943959
  64459348101831799681280613524058538734701140346391948
  43123602886715197650891859619406775380829934052483587
  24852816062664396208614460346356077862920426734827642
  39297480667395735991499439993352169042591437913726560
  87371539940176580242716026174273279118970901055428060
  07602652290786721234528626131896832489562551558690301
  69722949526329845768843465593188545343784216112898995
  32022728673678378918475624105043819026629471682080045
  96529501772278874463251649170353674520402106924748077
  58968397130026899411958043720556193694670060089356077
  85434235180939338357900691978526736 4,04,13,10,1,
  21616792031143752746309415579452320202510893126073087
  65238372340810506360007014776150093645457047086278697
  02069833686361660030089751732511247633740543213460
```

```
6
30354457021425165356037781636930036106404418295413726
64310025568369000670498679444999043128421557451025 43
72798191374275536426250198210571586202272343398573 81
39067694378476466514455677010796867090616789485577 45
86373708692617743377046549318417755362244202417503 90
31818274214414087856027953228184345511367502098318 42
52503669138492451882883596202775900911000502696129 43
37702970618450353224231935687225848095890069620605 87
42195460320178449784615826358214417743064260 3,65537,
22565864037247459832085566879900307360216368697408 73
38839648518084108748208582879216599979547434442126 25
88890147252456336915672583835216760904898276186048 82
34465993701091215757180129441242294617135912224358 33
08263282848559863033353276205119720064531420906643 44
66105512340849412585721082176775296236978626368464 19
04096127602145899784395121796324147878947439469952 71
60580771635153728361835033613801478959059921454822 59
18985677819917862619328741060567392938696268304282 17
31362106955244574788342442155479884548058065221901 71
09280678813179596281553889038339230703284433283787 26
38649586661550925176895542592715529562458882 3,65537,
42318605026103475553526985044230475639442568138229 96
39257340389895765698748645796695083715629551461265 47
87795529191443032330112839334617268576744268381873 89
67216638476720764074944647817540070989695696588210 95
10825583722503971588949457143192047749767090665830 95
96563894491959523986265780100040198815196631796286 63
10493446444221800190039054548479140813630569444720 84
76899929648797502449480345178716038904650329939041 65
59747823500160403807470368268041338870375750136244 46
80122222701317562224831854480644662034354542275281 22
73143870468255255703768242786844602119540704162618 23
57091188746940728386642358425545123156619 05,
77541428218992751486727644862498271472919514635342 58
31311913060905157240895568372058200945940190066173 81
78709143861531151462677050821533096659644045304151 77
74718735698700608641179259809996868147443542318468 89
69908468460498362071977017583852800075262462245389 69
09638240675638001652568590356741942911143120041676 23
56947622267619084456004720872031090989474470600876 82
```

```
6
64057292390739837356944925078045444352981697799440048112763789825376382626011524853438555391048758769078078923235520732259689515303259900452273102133197040813956548613688298554391758507875378779432847824073525835250217870849030986866114641442612783873
7 5,20,13,5,1,
24543526053222685920089002431387071080808753979168289339650707414109582770834242173009539282426745682722988055346700400544867910943785143909523891526583685873084778519448472887002660212182814126070863004440351914152540555901097499930235793743416545610368040032596055789846809140241500787296582367699342532180447446042851559187629052238168999819843726156363830030069139123051200441111945407276884741033640730462395861985143994176991609920694746571704707094532698910844357727419783667461542340930287506515793569704149384784131969168222943901917462927710491717935881329934336642631167278369058626398091411958822782486021,65537,225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676090489827618604882344659937010912157571801294412422946171359122243583308263282848559863033353276205119720064531420906643446610551234084941258572108217677529623697862636846419040961276021458997843951217963241478789474394699527160580771635153728361835033613801478959059921454822591898567781991786261932874106056739293869626830428217313621069552445747883424421554798845480580652219017109280678813179596281553889038339230703284433283787263864958666155092517689554259271552956245888 23,65537,17451642798421916838978706887752529294378563794943959644593481018317996812806135240585387347011403463919484312360288671519765089185961940677538082993405248358724852816062664396208614460346356077862920426734827642392974806673957359914994399933521690425914379137265608737153994017658024271602617427327911897090105542806007602652290786721234528626131896832489562551558690301697229495263298457688434655931885453437842161128989953202272867367837891847562410504381902662947168208004596529501772278874463251649170353674520402 10
```

```
7
  6924748077589683971300268994119580437205561936946700
  6008935607785434235180939338357900691978526 73,
  7754142821899275148672764486249827147291951463534258
  3131191306090515724089556837205820094594019006617381
  7870914386153115146267705082153309665964404530415177
  7471873569870060864117925980999686814744354231846889
  6990846846049836207197701758385280007526246224538969
  0963824067563800165256859035674194291114312004167623
  5694762226761908445600472087203109098947447060087682
  6405729239073983735694492507804544435298169779944004
  8112763789825376382626011524853438555391048758769078
  0789232355207322596895153032599004522731021331970408
  1395654861368829855439175850787537877943284782407352
  5835250217870849030986866114641442612783873
8 6,04,20,5,2,
  2161679203114375274630941557945232020251089312607308
  7652383723408105063600070147761500936454570470862786
  9702069833686361660030089751732511247633740543213460
  3035445702142516535603778163693003610640441829541372
  6431002556836900067049867944499043128421557451025 43
  7279819137427553642625019821057158620227234339857381
  3906769437847646651445567701079686709061678948557745
  8637370869261774337704654931841775536224420241750390
  3181827421441408785602795322818434551136750209831842
  5250366913849245188288359620277590091100050269612943
  3770297061845035322423193568722584809589006962060587
  4219546032017844978461582635821441774306426 03,65537,
  2454352605322268592008900243138707108080875397916828
  9339650707414109582770834242173009539282426745682722
  9880553467004005448679109437851439095238915265836858
  7308477851944847288700266021218281412607086300444035
  1914152540555901097499930235793743416545610368040032
  5960557898468091402415007872965823676993425321804474
  4604285155918762905223816899981984372615636383003006
  9139123051200441111945407276884741033640730462395861
  9851439941769916099206947465717047070945326989108443
  5772741978366746154234093028750651579356970414938478
  4131969168222943901917462927710491717935881329934336
  6426311672783690586263980914119588227824860 21,65537,
  4231860502610347555352698504423047563944256813822996
```

```
8
   392573403898957656987486457966950837156295514612654787795529191443032330112839334617268576744268381873896721663847672076407494464781754007098969569658821095108255837225039715889494571431920477497670906658309596563894491959523986265780100040198815196631796286631049344644422180019003905454847914081363056944472084768999296487975024494803451787160389046503299390416559747823500160403807470368268041338870375750136244468012222701317562224831854480644662034354542275281227314387046825525570376824278684460211954070416261823570911887469407283866423584255451231566190517451642798421916838978706887752529294378563794943959644593481018317996812806135240585387347011403463919484312360288671519765089185961940677538082993405248358724852816062664396208614460346356077862920426734827642392974806673957359914994399933521690425914379137265608737153994017658024271602617427327911897090105542806007602652290786721234528626131896832489562551558690301697229495263298457688434655931885453437842161128989953202272867367837891847562410504381902662947168208004596529501772278874463251649170353674520402106924748077589683971300268994119580437205561936946700600893560778543423518093933835790069197852673
9 7,04,13,10,2,65537,
   621616792031143752746309415579452320202510893126073087652383723408105063600070147761500936454570470862786970206983368636166003008975173251124763374054321346030354457021425165356037781636930036106404418295413726431002556836900067049867944499043128421557451025437279819137427553642625019821057158620227234339857381390676943784764665144556770107968670906167894855774586373708692617743377046549318417755362244202417503903181827421441408785602795322818434551136750209831842525036691384924518828835962027759009110005026961294337702970618450353224231935687225848095890069620605874219546032017844978461582635821441774306426035537,225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676090489827618604882344659937010912157571801294412422946171359122243583
```

```
 9
   0826328284855986303335327620511972006453142090664344
   6610551234084941258572108217677529623697862636846419
   0409612760214589978439512179632414787894743946995271
   6058077163515372836183503361380147895905992145482259
   1898567781991786261932874106056739293869626830428217
   3136210695524457478834244215547988454805806522190171
   0928067881317959628155388903833923070328443328378726
   3864958666155092517689554259271552956245888823,65537,
   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
    913912305120044111194540727688474103364073046239586 1
    985143994176991609920694746571704707094532698910844 3
    577274197836674615423409302875065157935697041493847 8
    413196916822294390191746292771049171793588132993433 6
    642631167278369058626398091411958822782486021,65537,
    225658640372474598320855668799003073602163686974087 3
    388396485180841087482085828792165999795474344421262 5
    888901472524563369156725838352167609048982761860488 2
    344659937010912157571801294412422946171359122243583 3
    082632828485598630333532762051197200645314209066434 4
    661055123408494125857210821767752962369786263684641 9
    040961276021458997843951217963241478789474394699527 1
    605807716351537283618350336138014789590599214548225 9
    189856778199178626193287410605673929386962683042821 7
    313621069552445747883424421554798845480580652219017 1
    092806788131795962815538890383392307032844332837872 6
    386495866615509251768955425927155295624588823,65537,
    174516427984219168389787068877525292943785637949439 5
    964459348101831799681280613524058538734701140346391 9
    484312360288671519765089185961940677538082993405248 3
    587248528160626643962086144603463560778629204267348 2
    764239297480667395735991499439993352169042591437913 7
    265608737153994017658024271602617427327911897090105 5
    428060076026522907867212345286261318968324895625515 5
    869030169722949526329845768843465593188545343784216 1
    128989953202272867367837891847562410504381902662947 1
    682080045965295017722788744632516491703536745204021 0
    692474807758968397130026899411958043720556193694670 0
    600893560778543423518093933835790069197852673,
    775414282189927514867276448624982714729195146353425 8
    313119130609051572408955683720582009459401900661738 1
    787091438615311514626770508215330966596440453041517 7
    747187356987006086411792598099968681474435423184688 9
    699084684604983620719770175838528000752624622453896 9
    096382406756380016525685903567419429111431200416762 3
    569476222676190844560047208720310909894744706008768 2
    640572923907398373569449250780454443529816977994400 4
    811276378982537638262601152485343855539104875876907 8
    078923235520732259689515303259900452273102133197040 8
    139565486136882985543917585078753787794328478240735 2
```

```
10 583525021787084903098686611464144261278387 3
11
```

```python
# coding=utf-8
import unittest

import src.simplify.flow_graph
import src.simplify.graph_objects
from src.crypto import keys
from src.transactions.ledger import *
from src.transactions.transaction import VerificationError


def setUpModule():
    print(os.getcwd())
    os.chdir("/home/tcassar/projects/settle")


def key_path(usr: str, keytype="private") -> str:
    assert usr == "d" or usr == "m" or usr == "t"
    return f"./src/crypto/sample_keys/{usr}_{keytype}-key.pe{'m' if keytype == 'private' else ''}"


class TestLedger(unittest.TestCase):
    def setUp(self) -> None:

        """Load in sample data from mock db"""

        self.valid: Ledger
        self.missing_key: Ledger
        self.invalid: Ledger

        # load in raw copies of d, m, t test keys
        ldr = keys.RSAKeyLoader()
        self.d_m_t_keys: dict[int, keys.RSAPrivateKey] = {}
        pubs: list[keys.RSAPublicKey] = []

        for person, id in zip(["d", "m", "t"], [4, 13, 20]):
            ldr.load(key_path(person))
            ldr.parse()
```

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

```
 73
 74         Check that first one goes through no issues
    , and that other two are caught
 75         """
 76
 77         with self.subTest("unsigned"), self.
    assertRaises(VerificationError):
 78             self.valid._verify_transactions()
 79
 80         # sign transactions in ledger; not normal
    procedure so contrived
 81         self.sign()
 82
 83         with self.subTest("signed"):
 84             self.valid._verify_transactions()
 85
 86         with self.subTest("missing key"), self.
    assertRaises(VerificationError):
 87             self.missing_key._verify_transactions()
 88
 89         with self.subTest("invalid key"), self.
    assertRaises(VerificationError):
 90             self.invalid._verify_transactions()
 91
 92     def test_as_flow(self):
 93         """Test that we build flow graphs from
    ledgers as expected"""
 94         # sign ledger
 95         self.sign()
 96
 97         Vertex = src.simplify.graph_objects.Vertex
 98
 99         exp = src.simplify.flow_graph.FlowGraph(
100             [Vertex(ID=4), Vertex(ID=13), Vertex(ID
    =20)]
101         )
102         d, m, t = exp.nodes()
103         exp.add_edge(d, (m, 10), (t, 5))
104         exp.add_edge(t, (m, 5))
105
106         as_flow = self.valid._as_flow()
```

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

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

```python
# coding=utf-8
import os
import unittest

from src.transactions.transaction import *


class TestTransaction(unittest.TestCase):
    def setUp(self) -> None:
        # load keys
        os.chdir("/home/tcassar/projects/settle/src")
        ldr = keys.RSAKeyLoader()
        ldr.load("./crypto/sample_keys/d_private-key.pem")
        ldr.parse()

        self.key = keys.RSAPrivateKey(ldr)
        self.pub_key = keys.RSAPublicKey(ldr)

        self.trn = Transaction(0, 0, 0, self.pub_key, self.pub_key)
        self.trn.time = 0  # type: ignore

    def test_hash(self):
        """Tests that calling .hash() on a Transaction displays expected behaviour"""
        self.assertEqual(
            self.trn.hash(),
            b"\xb9y\x8f\xe6\xdeo\x08\xd4\x1f\xdd\x01\xa3\xb4\xc5\x1d\x98\xd4\xac\x1bJ\x02\xd3\xa0\x01!\xf3\x99\xdf\xd5\\\x85v",
        )
        self.trn.time = 1
        self.assertNotEqual(
            self.trn.hash(),
            b"\xb9y\x8f\xe6\xdeo\x08\xd4\x1f\xdd\x01\xa3\xb4\xc5\x1d\x98\xd4\xac\x1bJ\x02\xd3\xa0\x01!\xf3\x99\xdf\xd5\\\x85v",
        )

```

```
34      def test_sign(self):
35          """
36          Checks that signing a Transaction displays
    expected behaviour
37          Working on assumption that rsa.Notary is
    working; tested in settle/tests/test_crypto"""
38
39          self.trn.sign(self.key, origin="src")
40
41          with (self.subTest("catch invalid origin"),
    self.assertRaises(ValueError)):
42              self.trn.sign(self.key, origin="no")
43
44          with self.subTest("invalid key types"), self
    .assertRaises(TransactionError):
45              self.trn.sign(12, origin="src")
46              self.trn.sign(self.pub_key, origin="dest
    ")
47
48          with self.subTest("sig_overwrite"), self.
    assertRaises(TransactionError):
49              self.trn.sign(self.key, origin="src")
50
51          with self.subTest("Right sig"):
52              self.assertEqual(
53                  b"\xc1~\xcbu\xecX\x01E*\xf2;O\xe3\
    xf3\x08x\xee\x84\xfc\xe1\xca\x8b\xa2\xedj\xec\x9b\
    xfd\xe5$7\x88n\xb7\x86\xfc~\x98\x91\x80Z\xcd\x1e\xf5
    }\xc2<JS\xefY\xe5UW\xfc\x0e\xd2\xbe\xc9\xea\xfbzm\
    xf2\xa7\x08\x8d\x05\x16\xe4@\xf40\x03I\xca\xbc.\x95\
    xbb\xd3\n\xfd9\xb0Wk\xf4\x03\x96\xdbF\xcd\xa0E\xd1\
    xac\xa6(\xb9\x92\xdb\x841\xe0U\"\xe4\x7f\xeb U\xc9Z\
    xe5\xf6\x19\xc1Wn\x17&\x17\x84Dt\xb6z\xc0\x02\x85`\
    xf3\xd3\xd5\x98t7\xcd_\xfc\xa7\\\xa7t\xe8\xb1\xafl\
    x05\xb3\x07a\xd0jr\xc4}\xd8\xb58\xf4o\x9f\x02\xa2\
    xe6?3B\xf6\xe1\xe6\xb2\x02d\xfd\xd5\x83\xcf\xcc\xb6\
    x06m\xc2p\xd7\x00@\x93H\xc6]\x0c\x98\x1e\xdf\xda\x00
    \x86\xd7\xec\xc7\x10\x025eiry\xd6\x80\xfe\xe2+\xb8\
    x1dn\noB\xc0\xa8a\xc8t\xbfg\xbb\xca(Aj\xa5\xeb\x94\
    x0c-\xacr'\xfe\n^Z\x13\xa0'\xaa\x96{\xf8x\xce\xd6\
    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
```