Testing

Test Coverage

In order to ensure that I have a robust technical solution, I have an extensive unit testing framework to cover the three most technically challenging areas of my program - the `transaction`, `crypto` and `simplify` modules. Across the three, I have 80% coverage, meeting the industry standard. The `crypto` module has >90% coverage. I will include the unit test results in this section. They were generated by my IDE and Python's `unittest` framework, so I could not associate them with my requirements exactly.

However, below the evidence of my tests passing is a link of unit tests to requirements. I have more unit tests than requirements as a byproduct of ensuring that my code is as robust as possible.

As is shown below, I was able to entirely complete my project, fulfilling all of my initial requirements.

Requirements

These are the same as 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 `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 of 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

    1. Digraphs
    2. Flow Graphs

4. Implementation of Edmonds-Karp

    1. Way to find 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. The server should be able to pull a group's transactions from a database, run the settling, and handle any requests from the client

4. The client should be able to request

    1. See their own user information
        1. Total debt across all groups
        2. Open transactions
        3. Closed transactions
    2. Open transactions / closed transactions
    3. Mark a transaction as settled
    4. Make / invite people / leave groups
    5. Settle a group
    6. Create a transaction
    7. Sign an open transaction
    8. Mark a transaction as settled

5. Ancillary functions should allow users to:

    1. Register for an account, giving email, name RSA private key in PEM format and a password
    2. A `whois` function, allowing you to see people's user info (name, email, public key)
    3. Create groups with a name and password
    4. Join groups by ID

Command Line Interface (D)

1. Everything listed in C.3

Database Architecture (E)

1. User information
    1. User ID
    2. Contact info
    3. Associated Groups
    4. Public Key
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 or not transaction has been settled
3. Group information

1. Group name
2. People in the group
3. Transactions in the group

Evidence of fulfilling A & B

: 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=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928426745682722988055346700400544867910943785143912
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216762
e=65537
---
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928426745682722988055346700400544867910943785143912
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216762
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928426745682722988055346700400544867910943785143912
e=65537
---
n=2161679203114375274630941557945232020251089312607308765238372340810506360007014776150093645457047086278697020698336863616600300897517325112
e=65537
n=2256586403724745983208556687990030736021636869740873388396485180841087482085828792165999795474344421262588890147252456336915672583835216762
e=65537
---
n=2454352605322268592008900243138707108080875397916828933965070741410958277083424217300953928426745682722988055346700400544867910943785143912
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

e=65537
n=225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676
e=65537
---
n=245435260532226859200890024313870710808087539791682893396507074141095827708342421730095392824267456827229880553467004005448679109437851439
e=65537
n=225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676
e=65537
---
n=216167920311437527463094155794523202025108931260730876523837234081050636000701477615009364545704708627869702069833686361660030089751732511
e=65537
n=245435260532226859200890024313870710808087539791682893396507074141095827708342421730095392824267456827229880553467004005448679109437851439
e=65537
---
n=65537,
e=621616792031143752746309415579452320202510893126073087652383723408105063600070147761500936454570470862786970206983368636166003008975173251
n=225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676
e=65537
---
n=245435260532226859200890024313870710808087539791682893396507074141095827708342421730095392824267456827229880553467004005448679109437851439
e=65537
n=225658640372474598320855668799003073602163686974087338839648518084108748208582879216599979547434442126258889014725245633691567258383521676
e=65537
---
verifying...
verified
verifying...
verified
verifying...
verified
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/pre_settle0'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'pre_settle0']
DEBUG:graphviz._tools:os.makedirs('./graph_renders')
DEBUG:graphviz.saving:write lines to './graph_renders/settled0'
DEBUG:graphviz.backend.execute:run [PosixPath('dot'), '-Kdot', '-Tsvg', '-O', 'settled0']

- **test_verify_transactions (Make three ledgers:)** 720 ms
  - [unsigned] passed
  - [signed] passed
  - [missing key] passed
  - [invalid key] passed

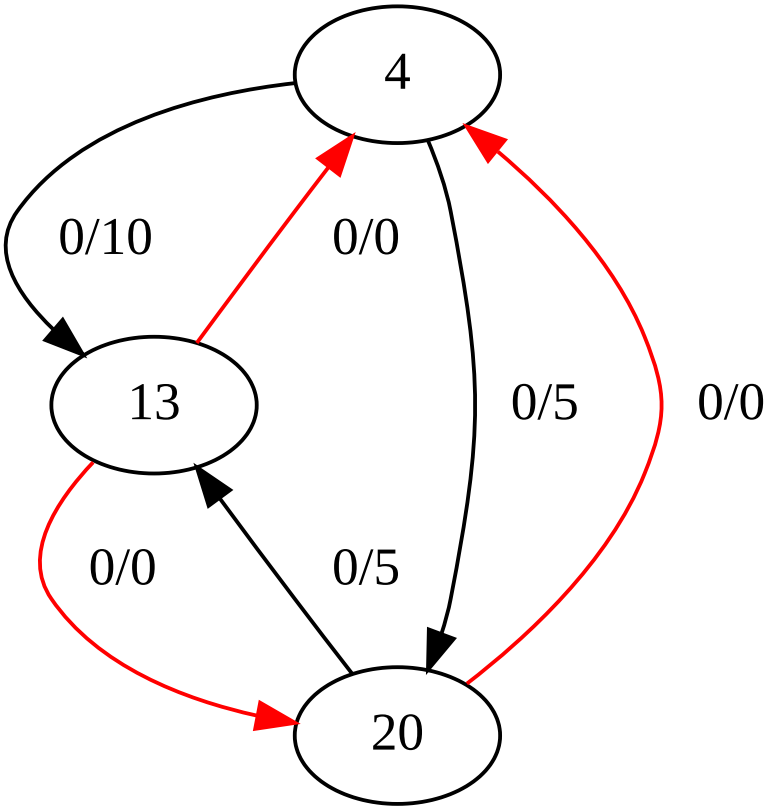
- **test_transaction** 488 ms
  - **TestTransaction** 488 ms
    - test_hash passed 170 ms
    - **test_sign (Working on assumption that rsa_Notary is working; tested in settle/tests/test_crypto)** 146 ms
      - [catch invalid origin] passed
      - [invalid key types] passed
      - [sig_overwrite] passed
      - [Right sig] passed
    - **test_verify** 172 ms
      - [catch invalid keys] passed
      - [good verif] passed
      - [priv/pub keys] passed
      - [verify src, dest] passed
      - [verify >1 param] passed
      - [bad key] passed

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

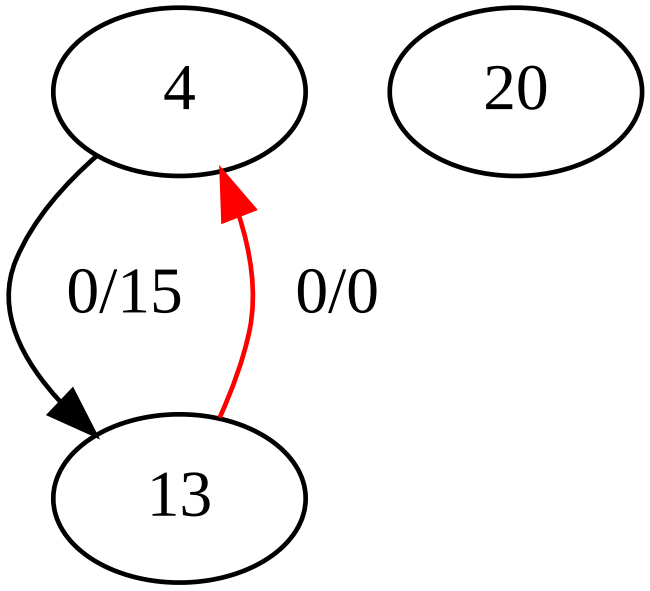
Hence, I am able to prove that not only have I achieved every requirement that I set out to under sections A & B, my extensive unit testing goes a long way to prove the robustness of my solution.

To show a visual proof of the correctness of my user-defined settling algorithm, I will provide some screenshots produced by test B5 (`test_flow.TestMaxFlow`). These graphs were generated during the running of the test - an artefact of the debugging process. However, I believe that they effectively show that I am able to simplify the chains of debt in a group.

Pre setting:



Post settling:



This is the same example system that I presented in my analysis section.

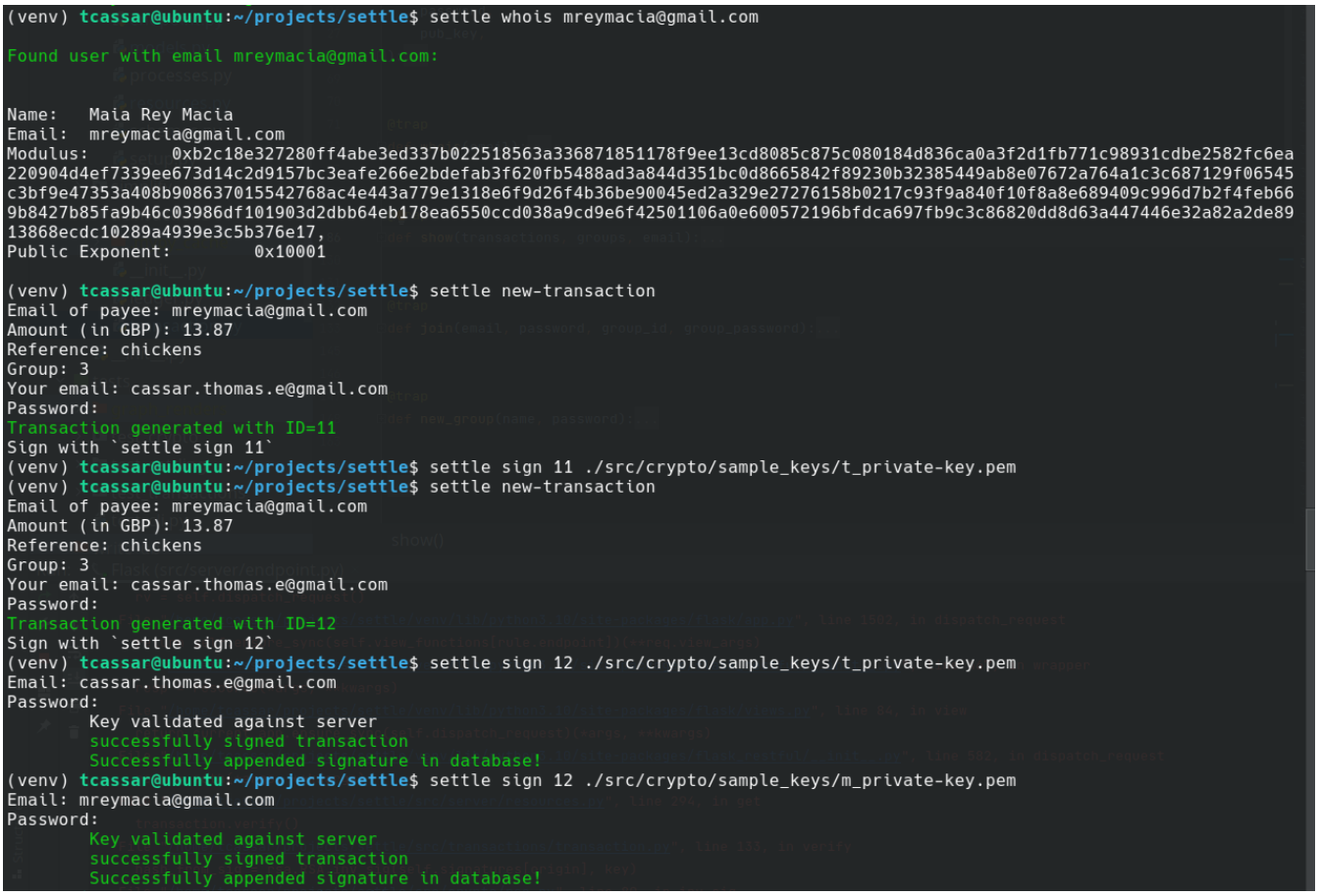
This is a debugging representation of a flow graph, hence the flows, capacities and residual edges (in red). However, it appropriately demonstrates that my technical solution is able to achieve its objective of simplifying groups of debt.

My unit tests, included at the end of this section, are more extensive than this one example. They are configured to check that flags are raised when the graph doesn't change, that it doesn't change when it shouldn't, and that the algorithm lends itself well to larger graphs.

During testing, I discovered that this algorithm was more effective on densely populated graphs (more on

this in the Evaluation). This solution ended up being an excellent heuristic model to solve my problem.

Evidence of fulfilling Section C & D



Evidence of fulfilling Section E

To show that I am able to store everything that I wanted to, as outlined in Section E, I will insert screenshots of my working database tables.

E1 - User information

USERS table (E1, E2, E3)

id	name	email	password	key_id
19	admin	admin@admin.com	b'\xfb\x00\x1d\xfc\xff\xd1\xc8\x99\xf3)xq@…	3
20	Tom Cassar	cassar.thomas.e@gmail.com	b'\xfb\x00\x1d\xfc\xff\xd1\xc8\x99\xf3)xq@…	4
21	Younes Tahir	keith@npl.com	b'\x90\xd9D\x8a6U\xc6<\x04\x92\x14Z\x86\xb…	5
22	foo	foo@bar.com	b'v\xd3\xbcA\xc9\xf5\x88\xf7\xfc\xd0\xd5\x…	6
23	Foo Bar	foobar@example.com	b'\t#H\x07\xe4\xaf\x85\xf1|f\xb4\x8e\xe3\x…	7
24	Maia Rey Macia	mreymacia@gmail.com	b'8\xdd\x08\x8b5S\x81>\x8e0p/\xf6\x16o(\x9…	8
25	Kez Carey	kezza@cherryactive.com	b'\xff\x8es\xe7\xb3\x1f\x12\x1e\xeb\xbe46w…	9
26	adhish	adhish@gmail.com	b'\xf5\x1a\xfe\x17\xc9\xe1\x85\xc9\xce\xb9…	10

GROUP_LINK table (E4)

	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

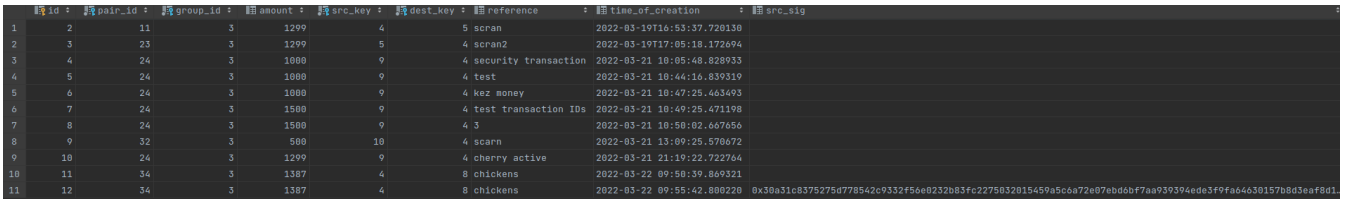
KEYS table (E5)

	id	n	e
1	3	0xc26c13c82f5df38ebedf77256144a471dfb1f62f…	0x10001
2	4	0xc26c13c82f5df38ebedf77256144a471dfb1f62f…	0x10001
3	5	0xb2c18e327280ff4abe3ed337b022518563a33687…	0x10001
4	6	0xb2c18e327280ff4abe3ed337b022518563a33687…	0x10001
5	7	0xc26c13c82f5df38ebedf77256144a471dfb1f62f…	0x10001
6	8	0xb2c18e327280ff4abe3ed337b022518563a33687…	0x10001
7	9	0xab3cec18ae8df68c502ed71ce3376cca00c5e82a…	0x10001
8	10	0xc26c13c82f5df38ebedf77256144a471dfb1f62f…	0x10001

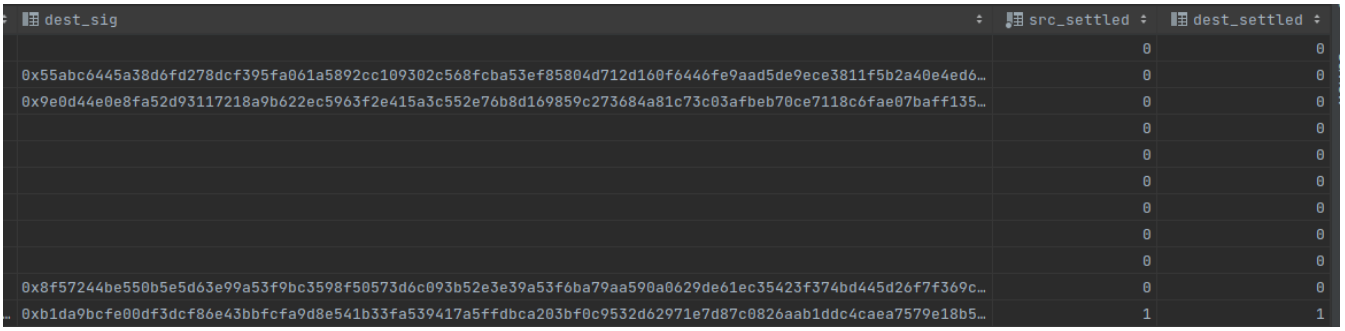
(note: all the public exponents (e) are the same. This is not an error - RSA guidelines state that the public exponent generated with keys is normally 65537 by convention. In hex, this is $0x10001$).
While no users here have multiple public keys, all key information has been normalised out to allow for this in the future while maintaining database integrity and a normalised structure

E2 - Transaction Information

The following two screenshots provides evidence for E2.1, E2.4 → E2.8



	id	pair_id	group_id	amount	src_key	dest_key	reference	time_of_creation	src_sig
1	2	11	3	1299	4	5	scran	2022-03-19T16:53:37.720130	
2	3	23	3	1299	5	4	scran2	2022-03-19T17:05:18.172694	
3	4	24	3	1000	9	4	security transaction	2022-03-21 10:05:48.828933	
4	5	24	3	1000	9	4	test	2022-03-21 10:44:16.839319	
5	6	24	3	1000	9	4	kez money	2022-03-21 10:47:25.463693	
6	7	24	3	1500	9	4	test transaction IDs	2022-03-21 10:49:25.471198	
7	8	24	3	1500	9	4	3	2022-03-21 10:50:02.667656	
8	9	32	3	500	10	4	scarn	2022-03-21 13:09:25.570672	
9	10	24	3	1299	9	4	cherry active	2022-03-21 21:19:22.722764	
10	11	34	3	1387	4	8	chickens	2022-03-22 09:50:39.869321	
11	12	34	3	1387	4	8	chickens	2022-03-22 09:55:42.800220	0x30a31c8375275d778542c9332f56e0232b83fc2275032015459a5c6a72e07ebd6bf7aa939394ede3f9fa64630157b8d3eaf8d1



dest_sig	src_settled	dest_settled
	0	0
0x55abc6445a38d6fd278dcf395fa061a5892cc109302c568fcba53ef85804d712d160f6446fe9aad5de9ece3811f5b2a40e4ed6…	0	0
0x9e0d44e0e8fa52d93117218a9b622ec5963f2e415a3c552e76b8d169859c273684a81c73c03afbeb70ce7118c6fae07baff135…	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
	0	0
0x8f57244be550b5e5d63e99a53f9bc3598f50573d6c093b52e3e39a53f6ba79aa590a0629de61ec35423f374bd445d26f7f369c…	0	0
0xb1da9bcfe00df3dcf86e43bbfcfa9d8e541b33fa539417a5ffdbca203bf0c9532d62971e7d87c0826aab1ddc4caea7579e18b5…	1	1

The evidence for E2.2 and E2.3 is provided below

	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

This is a link table associating pairs of people together (order of a pair matters, i.e. the row 20, 21 is different from 21, 20)
This was done to allow pairs of people to have multiple transactions, maintaining a normalised structure and ensuring data integrity.

E3 - Group Informaton

Groups Table: This satisfies require E3.1, E3.2

	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'

Group Link Table: This satisfies E3.3

	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

Requirement E3.4 is satisfied by this column in the transactions table

group_id
3
3
3
3
3
3
3
3
3
3
3

Thus, the entirety of my requirements under Section E have been achieved.

Test code source: