



## Blockchain for the Next Generation Internet



---

## D4 . "PROTOTYPE DEMONSTRATION" FULL DESIGN SPECIFICATION

---

**REPUTABLE**

07/10/2021



Grant Agreement No.: 957338

Call: H2020-ICT-2020-1

Topic: ICT-54-2020

Type of action: RIA

## REPUTABLE

### D4 . "PROTOTYPE DEMONSTRATION" FULL DESIGN SPECIFICATION

DUE DATE	08/10/2021
SUBMISSION DATE	08/10/2021
TEAM	Junaïd Arshad, M. Ajmal Azad, Alousseynou Prince, Jahid Ali, Bibek Khattri
VERSION	1.0
AUTHORS	Junaïd Arshad, M. Ajmal Azad, Alousseynou Prince, Jahid Ali, Bibek Khattri



---

## EXECUTIVE SUMMARY

---

The ONTOCHAIN initiative emphasizes trustworthy information exchange which requires a reliable and privacy-preserving reputation system. Through the state of the art analysis conducted as part of D1 of this project, we identified the fundamental requirements for an effective reputation system for ONTOCHAIN. These require a privacy-preserving, decentralised and verifiable reputation system which is able to preserve provenance of reputation information and the reputation scores deduced from this information. Further, the reputation data should be queryable to facilitate user verification as well as seamless integration with systems which may envisage leveraging such information to deliver trustworthy services.

REPUTABLE addresses these requirements and has the potential to deliver a cross-platform privacy-aware reputation system which leverages blockchain technology to achieve decentralised, verifiable calculation of reputation scores. Further it enables interaction with end users and systems through a secure, reputation analytics dashboard to facilitate user verification as seamless integration with other systems and services.

This document presents a detailed insight into the implementation of the REPUTABLE system in-line with the design specifications presented in D3. It includes description of different components, their implementation details, and details on how to use them.

## TABLE OF CONTENTS

1	TECHNICAL RESULTS .....	8
1.1	PROJECT REPOSITORIES .....	8
1.2	PROGRAM MODULES OVERVIEW .....	8
1.3	APIS FOR SDKS .....	9
1.4	RESTAPIS FOR SERVICES .....	9
1.4.1	Aggregator .....	10
1.5	ONTOLOGIES .....	11
1.6	HOW TO RUN THE CODE .....	11
1.6.1	Web Interfaces (Marketplace, Feedback & Dashboard) .....	11
1.6.1.1	The contents of the repository .....	11
1.6.1.2	How to Compile .....	12
1.6.1.3	How to Deploy .....	13
1.6.2	Proprietary Oracle .....	17
1.6.3	Gateway Smart Contract .....	20
1.6.4	Aggregator module .....	21
1.7	HOW TO TEST .....	26
1.7.1	Tests for Web Interfaces .....	26
1.7.2	Tests for Aggregator Module .....	28
2	TECHNICAL VALUE ADDED .....	30
2.1	KEY INNOVATIONS OF YOUR SOLUTION, IN SUMMARY .....	30
2.2	EVENTUAL SUGGESTED EVOLUTIONS OF YOUR SOLUTION .....	30
2.3	PRESENT OR FUTURE PATENTABILITY OF YOUR SOLUTION .....	31
2.4	TECHNICAL KPIS .....	31
2.4.1	INTEROPERABILITY AND STANDARDIZATION .....	32
2.4.2	KPIS TOWARDS INNOVATION .....	32
2.4.3	KPIS TOWARDS MORE HUMAN-CENTRIC EVOLUTION OF THE INTERNET ..	32
2.4.4	KPIS TOWARDS MORE DECENTRALIZED NGI .....	33
2.4.5	KPIS TOWARDS NEW FORMS OF INTERACTION AND IMMERSIVE ENVIRONMENTS FOR NGI USERS .....	34
2.4.6	KPIS RELATED TO THE IMPLEMENTATION .....	35

3	COMMUNICATION, DISSEMINATION, EXPLOITATION .....	36
3.1	YOUR CONTENTS .....	36
3.1.1	MAIN ACHIEVEMENTS .....	36
3.1.2	DEMO .....	36
3.1.3	YOUR TESTIMONIAL .....	37
3.2	COMMUNICATION, DISSEMINATION, EXPLOITATION RESULTS .....	37
3.2.1	NEWS .....	37
3.2.2	PRESS RELEASES .....	37
3.2.3	SCIENTIFIC PAPERS .....	37
3.2.4	EXHIBITIONS .....	37
4	APPENDIX: FINAL DESIGN UPDATES .....	38

## LIST OF FIGURES

FIGURE 1: THE CONTENTS OF THE DIRECTORY.....	12
FIGURE 2: VISUALSTUDIO CODE VERSION NUMBER.....	12
FIGURE 3: THE COMPATIBLE DEPENDENCIES AT THE POINT OF DEVELOPMENT.....	13
FIGURE 4: THE WEB APPLICATION MAIN PAGE.....	14
FIGURE 5: ADDING ITEMS TO CART.....	14
FIGURE 6: MESSAGE SHOWN UPON SUCCESSFULLY PURCHASING ITEMS.....	15
FIGURE 7: EMAIL RECEIVED WITH LINKS FOR FEEDBACKS.....	15
FIGURE 8: THE FEEDBACK PAGE.....	16
FIGURE 9: USER DASHBOARD.....	16
FIGURE 10: USE OF FIREBASE DB AND GRAPHICAL VISUALISATION OF DATA STORAGE.....	17
FIGURE 11 - FEEDBACK.JS VARIABLE HOLDING THE ACCOUNT AND CONTRACT ADDRESS.....	21
FIGURE 12 - ORACLE PYTHON CODE WITH VARIABLES FOR SMART CONTRACT ADDRESSES.....	22
FIGURE 13 - TRUFFLE COMPILE COMMAND AND OUTPUT.....	23
FIGURE 14 - DEPLOYMENT OF CONTRACT TO GANACHE USING REMIX WEB3 PROVIDER.....	24
FIGURE 15 - MODAL CONFIRMATION TO NOTIFY THAT THE SCORE HAS SUCCESSFULLY BEEN ADDED TO THE BLOCKCHAIN.....	25

## ABBREVIATIONS

<b>OC1</b>	Open Call 1
<b>PoS</b>	Proof of Stake
<b>PoW</b>	Proof of Work
<b>PoR</b>	Proof of Reputation
<b>REPUTABLE</b>	A Provenance-aware Decentralized Reputation System for Blockchain-based Ecosystems
<b>RDOS</b>	Reputation Data Oracle



## 1 TECHNICAL RESULTS

### 1.1 PROJECT REPOSITORIES

**Hint:** your software is typically released on the ONTOCHAIN private repository. In this case identify restrictions before applying to the public. Some projects have also released their software on a public repository: in this case applies, adds it in the table

REPOSITORY NAME	Link	Kind	Eventual restrictions
GitHub	<a href="https://github.com/ONTOCHAIN/REPUTABLE">https://github.com/ONTOCHAIN/REPUTABLE</a>	ONTOCHAIN	
GitHub	<a href="https://github.com/rjarshad/REPUTABLE">https://github.com/rjarshad/REPUTABLE</a>	Private	TBC

### 1.2 PROGRAM MODULES OVERVIEW

**Hint:** separate your implementation in modules, is applicable, and fill the following summary table.

SDKs are detailed in the paragraph 1.3, with their APIs, services are detailed in the paragraph 1.4 with their APIs.

Name	Description	Language	Kind
Dummy marketplace and user feedback interface	We have created a dummy marketplace to simulate a scenario where user can buy things and then provide feedback about the seller through a link	React	Service

	received in the email.		
<b>User Feedback Smart Contract</b>	The module gathers feedback provided by individual users and stores them on the chain	Solidity	Service
<b>Proprietary Oracle</b>	It acts as a bridge between feedback SC and the aggregator	python	Service
<b>Aggregator</b>	The aggregator calculates an aggregate score using the individual user feedbacks	Python	REST Service
<b>Gateway contract</b>	Stores: aggregate feedback to the blockchain, and individual feedback to an off-chain storage	Solidity	Service
<b>Dashboard</b>	Provides the ability for users to query seller reputation score, verify seller reputation score, and query individual user feedback	React	Service

### 1.3 APIS FOR SDKS

**Hint:** describe here only the SDK modules at the API level. If your implementation does not include SDK modules, simply write the sentence "No SDK in this implementation", without deleting this paragraph, to maintain aligned the numbers of your document.

No SDK in this implementation

### 1.4 RESTAPIS FOR SERVICES

**Hint:** describe here only the RESTAPIs for your services. If your implementation does not include services, simply write the sentence "No

Services in this implementation", without deleting this paragraph, to maintain aligned the numbers of your document.

#### 1.4.1 Aggregator

##### Description:

This module is responsible to calculate aggregate reputation score using the individual user feedback. The aggregator is a crucial component of the REPUTABLE system

HTTP method	URI	Arguments	Return value	Description
GET	/reputation	sellerId (int)	Seller reputation score (int)	To query aggregate reputation score for a seller
POST	/reputation_score	sellerId (int), Indiv reputation scores (array)	Seller reputation score (json)	Post seller aggregate score using seller id and user scores
GET	/verify_reputation	sellerId (int)	Receipt highlighting on-chain record/hash of reputation feedback	to verify reputation of a service/seller
GET	/individual_score	sellerId (int), userId(int)	the individual score of a buyer that was used to rate the seller	get individual score of a buyer/consumer

GET	/individual_scores	sellerId(int)	the individual scores of buyers who rated the specified seller (array)	get individual scores relating to a buyer's aggregated score
GET	/token_used	sellerId(int), token (int)	Boolean that confirms whether or not a token has been used before	query to find out if a token has previously been used

## 1.5 ONTOLOGIES

**Hint:** If your implementation does not include ontologies, simply write the sentence "No ontology in this implementation", without deleting this paragraph. Mention first ontologies you have specifically developed for the ONTOCHAIN project (new), then ontologies you have already developed before ONTOCHAIN, and you updated for the ONTOCHAIN project (MyUpdated) or simply reused (MyReused). Then mention eventual third-party ontologies you simply reused, e.g., Good Relations.

"No ontology in this implementation".

## 1.6 HOW TO RUN THE CODE

As our software includes different components which have specific dependencies, we describe them individually in the interest of readers' understanding.

### 1.6.1 Web Interfaces (Marketplace, Feedback & Dashboard)

#### 1.6.1.1 The contents of the repository

The repo contains various directories as shown on **Figure [1]**. These are all required to run the application. Src > pages contain the main pages (screens) for the web application.

images	21/09/2021 22:45	File folder	
node_modules	29/09/2021 15:31	File folder	
public	18/09/2021 21:31	File folder	
src	26/09/2021 17:26	File folder	
.gitignore	13/09/2021 15:07	Text Document	1 KB
package.json	29/09/2021 15:31	JSON File	2 KB
package-lock.json	29/09/2021 15:31	JSON File	763 KB
README.md	24/09/2021 18:17	MD File	1 KB
yarn.lock	18/09/2021 21:31	LOCK File	490 KB

FIGURE 1: THE CONTENTS OF THE DIRECTORY.

### 1.6.1.2 How to Compile

The web application has been developed and can be deployed through VISUAL STUDIO CODE version 1.60.2, code editor. A more detailed info about the version of VS code is shown on Figure 2.

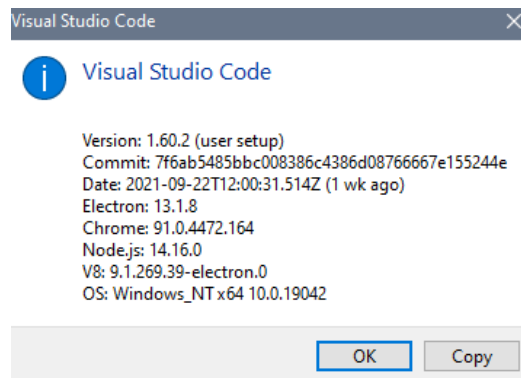


FIGURE 2: VISUALSTUDIO CODE VERSION NUMBER

Upon opening the source folder with VS code, you are to run "npm install". To do this navigate to the top toolbar select "terminal" > "new terminal" > this will open the terminal interface then, navigate into your application directory. From here you can run the command "npm install".

```
"dependencies": {
  "@emotion/react": "^11.4.1",
  "@emotion/styled": "^11.3.0",
  "@material-ui/core": "^5.0.0-beta.5",
  "@material-ui/styles": "^4.11.4",
  "@material/dialog": "^12.0.0",
  "@types/react-autosuggest": "^10.1.5",
  "bootstrap": "^5.1.0",
  "cra-template": "1.1.2",
  "emailjs-com": "^3.2.0",
  "firebase": "^9.1.0",
  "material-ui": "^1.0.0-beta.47",
  "react": "^17.0.2",
  "react-bootstrap": "^2.0.0-beta.6",
  "react-dom": "^17.0.2",
  "react-dropdown-input": "^0.1.11",
  "react-router-dom": "^5.3.0",
  "react-scripts": "4.0.3",
  "react-use-cart": "^1.11.2",
  "reactstrap": "^8.10.0",
  "simple-random-number-generator": "^1.2.0"
},
```

FIGURE 3: THE COMPATIBLE DEPENDENCIES AT THE POINT OF DEVELOPMENT

#### Compatible Dependencies:

This command will locally install all the dependencies and packages required to run the web application. If you are to encounter a package related error upon deployment you will need to upgrade or downgrade to the compatible packages used during the development of the application as shown on Figure 3.

Upon successfully running **"npm install"** without any errors, the application will be compiled and ready to be deployed.

#### 1.6.1.3 How to Deploy

Using the Terminal opened on VS Code enter **"npm run"** this will deploy the web application locally. An internet browser should open with the link of <http://localhost:3000/> and the store which is called the fruit shop. This contains generic functionality of online shopping store **Figure 4**.

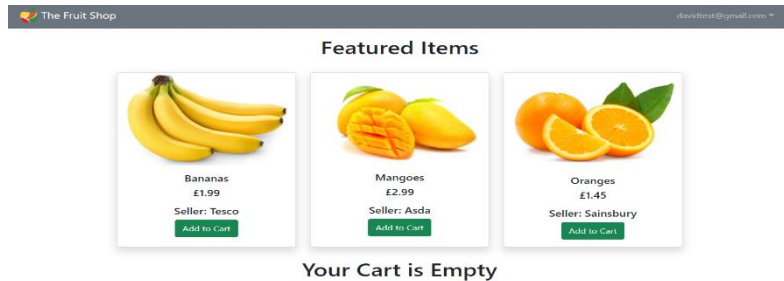


FIGURE 4: THE WEB APPLICATION MAIN PAGE

### How to Purchase Items and get an Email:

To aid the ease explanation of the functionality, the web interface is embedded with an email notification which gets triggered when the user buys from the store. The email is received on this test account (Email: **reputable.blockchain@gmail.com**, Password: **NGIReputable786**). You are to log in into to view the emails.

Firstly, add items to the cart, using the "Add to Cart Buttons" provided on the store, as shown on **Figure 5**.

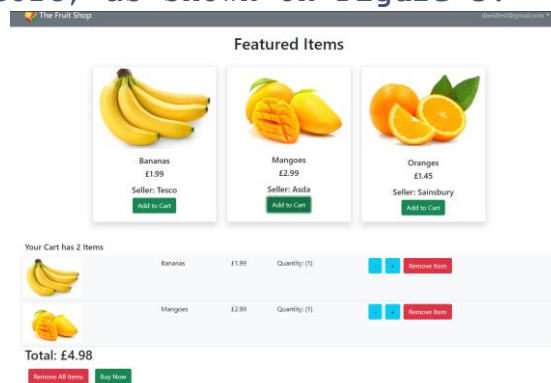


FIGURE 5: ADDING ITEMS TO CART

Secondly, when you have the required items in the cart simply click "buy now", this will provide user message as shown on **Figure 6**.



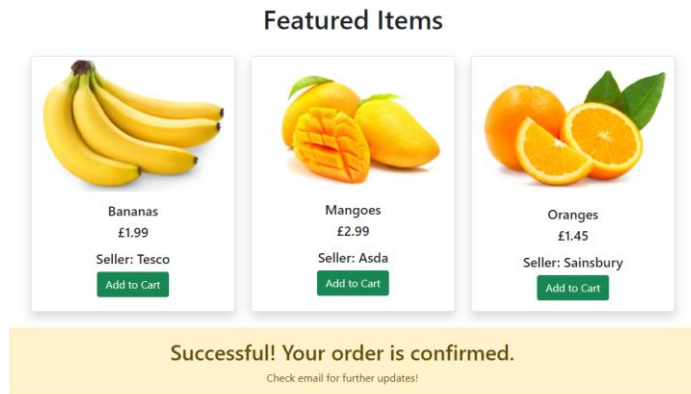


FIGURE 6: MESSAGE SHOWN UPON SUCCESSFULLY PURCHASING ITEMS

Last but not least, you will receive an email, in the provided Gmail account as per the scenario. The content of the email will have links where the user is able to provide feedback of various sellers.

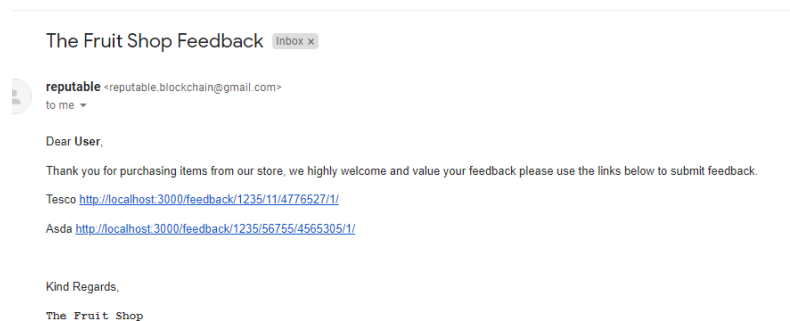


FIGURE 7: EMAIL RECEIVED WITH LINKS FOR FEEDBACKS

### How to provide feedback:

Buyers can provide feedback through the feedback links which are received as shown on Figure 7, simply click on the link of the seller and it will open a new browser which contains information a question as shown on Figure 8.



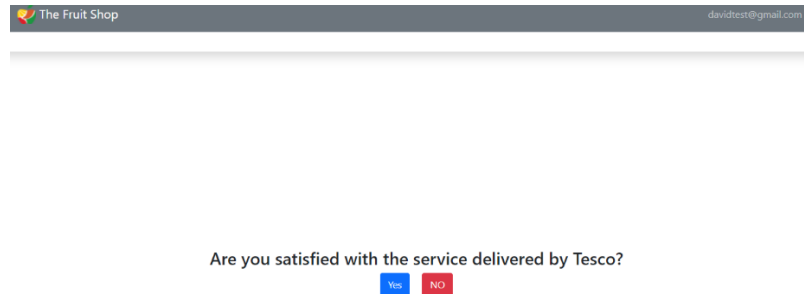


FIGURE 8: THE FEEDBACK PAGE

Simply, provide the answer by press **Yes/NO** which will be further processed into the other systems. A pop of model will be presented notifying that the user that the feedback has been successfully submitted.

#### Accessing the Dashboard:

Dashboard contains the functionality to perform various queries. It can be accessed using this link <http://localhost:3000/dashboard>.

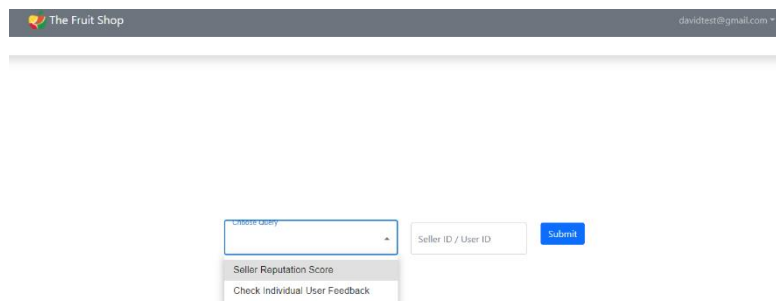


FIGURE 9: USER DASHBOARD

#### Cloud Storage using Firebase Firestore DB:

Using the Gmail account provided (Email: **reputable.blockchain@gmail.com**, Password: **NGIReputable786**), the google firebase console is available,

this is where the document of individual scores are stored and retrieved to be used in the Web Application

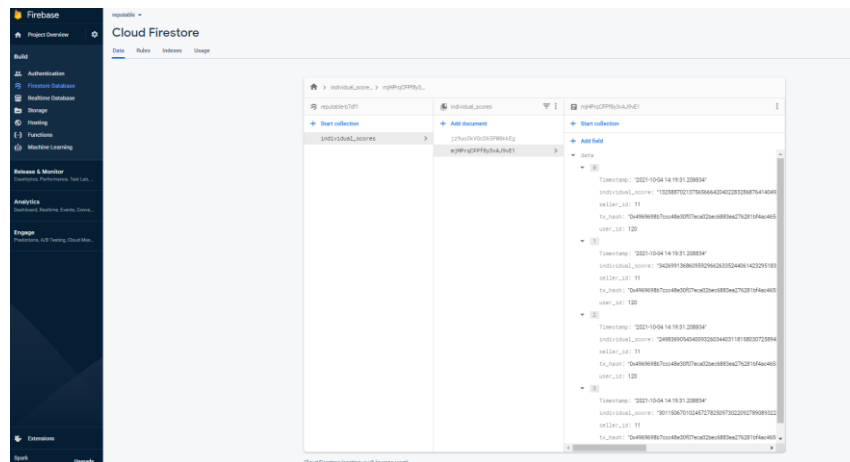


FIGURE 10: USE OF FIREBASE DB AND GRAPHICAL VISUALISATION OF DATA STORAGE

## 1.6.2 Proprietary Oracle

### Requirements and Dependencies:

Python 3.x

web3.py 5.24.0

python-paillier 1.4.0

firebase-admin 5.0.3

Remix IDE (<https://remix.ethereum.org/>)

Windows 10

### Functionality of proprietary oracle:

The proprietary oracle consists of an oracle smart contract and an off-chain Python backend that serve the core functions of the oracle.

This backend is responsible for performing the off-chain computation and sending the results back into the smart contract(s). This is done by utilizing the Web3.py library to listen for events emitted by the oracle smart contract and perform the necessary operation depending on which event was emitted. The oracle also establishes a connection to the cloud to also store some of the results off-chain.

The oracle smart contract (oracle.sol) is responsible for sending a request and any required parameters to the off-chain Python oracle. To accomplish this, this smart contract will emit an event depending on the service that the off-chain oracle is requested to do. There are two services that the off-chain oracle currently supports and that this smart contract will generate an event for:

- Encryption of individual score - This would emit the 'RequestScoreEvent' which takes the parameters 'sellerId' (uint) and 'indi\_score' (uint). The results of each individual score would also be stored on Cloud Firestore
- Aggregation of encrypted scores - This would emit the 'RequestValueEvent' which takes the parameters 'campaignId' (uint), 'sellerId' (uint), 'userId' (uint) and 'array' uint[]. The results of each aggregation would also be stored on Cloud Firestore

If the aggregation service was chosen, then the aggregate score would be returned to the 'gateway' contract (gateway.sol) via the 'returnToGateway' function. This function creates an interface to interact with the gateway contract. The transaction hash generated from the off-chain oracle interacting with this function is also stored on the cloud database.

The address for the deployed oracle smart contract must be entered for the variable 'oracle\_address'

#### Oracle Smart Contract:

Function	Input	Output
<b>requestValue</b>	uint _campaign_id, uint _sellerId, uint _userId, uint array array	Make request to off-chain oracle to calculate aggregate score
<b>requestScore</b>	uint _sellerId, uint _indi_score	Make request to off-chain oracle to encrypt individual score
<b>returnToGateway</b>	Address _gateway_address, uint _aggr_score, string _off_chain	Return encrypted aggregated score to gateway smart contract

## Off-chain Oracle:

Function	Input	Output
<b>aggregate</b>	seller_id, ind_data	Calculate an aggregate score from the array of encrypted numbers provided in 'ind_data'
<b>encrypt_score</b>	score	Encrypt the value stored in the variable 'score'
<b>handle_event</b>	event	Processes the event to gather event and parameters. This is responsible for determining which oracle service is performed i.e., encrypting individual score or aggregation. These values are also stored onto the cloud
<b>log_loop</b>	event_filter, poll_interval	Regularly polls to see if there are new events depending on the configuration of the event filter. The poll interval determines how long execution is paused before polling for new events

## How to deploy:

"python3 oracle.py"

### 1.6.3 Gateway Smart Contract

#### Dependencies

firebase-admin 5.0.3

Remix IDE (<https://remix.ethereum.org/>)

#### Functionality of gateway contract:

Once the aggregate score for a seller has been calculated off-chain, the result is then stored on-chain via the gateway smart contract (gateway.sol). The gateway contract contains a 'callback' function to which the oracle smart contract is able to interact with to send values which it itself has received from the off-chain oracle.

The callback functions contains two parameters:

- `_aggr_score` - This contains the aggregate score calculated in the off-chain oracle. Due to its length this variable is stored as a string
- `_off_chain` - This contains the location of the cloud database (in this instance Cloud Firestore) JSON file of all the individual scores used in the aggregation with its transaction hash

This contract also contains the functions "get\_on\_chain" and "get\_off\_chain" which retrieve the aggregate score stored on-chain and cloud database URL respectively.

The address for the deployed oracle smart contract must be entered for the variable 'gateway\_address'.

#### Gateway Contract:

Function	Input	Output
<code>get_on_chain</code>	No input	Calculated encrypted aggregate score
<code>get_off_chain</code>	No input	Location of cloud database stored online

<b>callback</b>	string _aggr_score, string _off_chain	Calculated aggregate score stored in variable 'aggr_score' and cloud location stored on 'off_chain'
-----------------	---	---

### How to deploy:

To deploy smart contracts, use Remix IDE

## 1.6.4 Aggregator module

### How to run the code:

Install dependencies for the oracle and the swagger API by running "pip install -r requirements.txt".

Run the ganache blockchain and compile the smart contracts using the "truffle compile" command. Deploy the smart contract on ganache using remix's web3 provider.

Get the address of the deployed data service smart contract and set it on the feedback web\_address variable.

Take the first account on ganache and set it as the variable address on feedback.js (see fig. 11).

```

const Feedback = () => {
  const location = useLocation();
  const storeName = location.state?.fromDashboard;
  console.log(storeName);
  const userName = "David";
  const history = useHistory();
  const [open, setOpen] = React.useState(false);
  const [tokenUsedModal, setTokenUsedModal] = React.useState(false);
  const { userID, sellerID, campaignID, tokenID } = useParams();
  const [sName, setSName] = React.useState("");
  const address = '0x6148918d867c92f7ada56Cbd99Be60F74353904c';
  const web_address = '0xEb132f9B8142630Eb855B31D046FE2518F77fdEF';

```

FIGURE 111 - FEEDBACK.JS VARIABLE HOLDING THE ACCOUNT AND CONTRACT ADDRESS

As shown on figure 12, set the address of the data service, oracle, gateway and on-chain reputation data smart contract on oracle.py and SwaggerAPI.py.

```
blockchain_address = 'HTTP://127.0.0.1:8545'
# Client instance to interact with the blockchain
web3 = Web3(HTTPProvider(blockchain_address))
# Set the default account (so we don't need to set the "from" for every transaction call)
web3.eth.defaultAccount = web3.eth.accounts[0]

oracle_compiled_path = './src/abi/OracleInterface.json'
oracle_address = '0x496AEa23C9aFFBaFDeb182f65C908b0f1105FeE'
with open(oracle_compiled_path) as file:
    oracle_json = json.load(file) # load contract info as JSON
    oracle_abi = oracle_json['abi']
    #print("oracle_abi: ", oracle_abi)
oracle_contract = web3.eth.contract(address=oracle_address, abi=oracle_abi)
#getSellerId()

gateway_compiled_path = './src/abi/GatewayInterface.json'
gateway_address = '0x27f2F602e60cb45158f0A7177875c0363a36c8C5'
with open(gateway_compiled_path) as file:
    gateway_json = json.load(file) # load contract info as JSON
    gateway_abi = gateway_json['abi']
gateway_contract = web3.eth.contract(address=gateway_address, abi=gateway_abi)

onchain_compiled_path = './src/abi/OnchainReputationData.json'
onchain_address = '0x98444A0988eA808d7eDb5Ce43630e99ED93CB579'
with open(onchain_compiled_path) as file:
    onchain_json = json.load(file) # load contract info as JSON
    onchain_abi = onchain_json['abi']
onchain_contract = web3.eth.contract(address=onchain_address, abi=onchain_abi)

web_compiled_path = './src/abi/WebInterface.json'
web_address = '0xEb132f9B8142630Eb855B31D046FE2518F77fDEF'
with open(web_compiled_path) as file:
    web_json = json.load(file) # load contract info as JSON
    web_abi = web_json['abi']
web_contract = web3.eth.contract(address=web_address, abi=web_abi)
```

FIGURE 112 - ORACLE PYTHON CODE WITH VARIABLES FOR SMART CONTRACT ADDRESSES

Launch oracle.py and swaggerAPI.py each on a console terminal.

Run "npm start" on visual studio terminal to launch the front-end web interface.

Buy an item from the shop to get a link through which to rate sellers.

To view the dashboard, visit localhost:3000/dashboard.

For the swagger API, visit localhost:5000/docs for experimenting with the endpoints of the API.



## How to Compile

Compiling the contracts on windows 10. Libraries such as truffle, web3, emailjs, etc can be installed for the front-end web application using "npm install".

Compilation of the smart contracts is done using truffle. The command "truffle compile" on visual studio's terminal will compile all the solidity files in the contracts folder to create json files of the metadata. The output directory of these json files can be defined in the truffle config file.

```
testUI1 (prince)
$ truffle compile

Compiling your contracts...
=====
> Compiling .\contracts\DataService.sol
> Compiling .\contracts\Gateway.sol
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\OnchainReputationData.sol
> Compiling .\contracts\Oracle.sol
> Compilation warnings encountered:
```

FIGURE 113 - TRUFFLE COMPILE COMMAND AND OUTPUT

Aside from compiling smart contracts using truffle as shown on figure 13, the smart contracts can also be compiled using remix. This will however require copying the compiled files to the web application directory.

## How to Deploy

Deploying the smart contract is done using remix (<https://remix.ethereum.org>) on ganache, a personal blockchain. Web3 provider is chosen as the environment when selecting the network to deploy and if ganache is already running, remix will identify the ethereum node via HTTP as shown on figure 14.



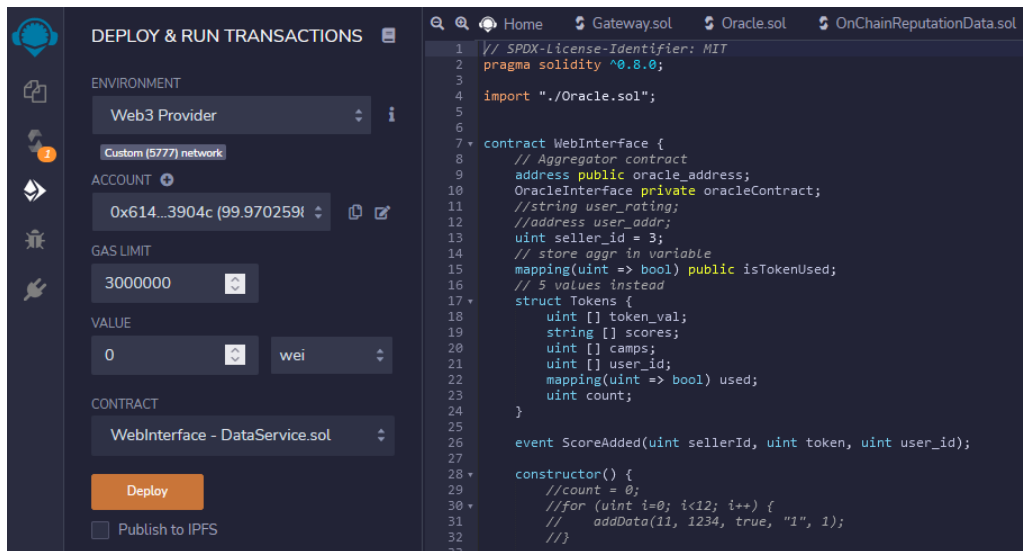


FIGURE 114 – DEPLOYMENT OF CONTRACT TO GANACHE USING REMIX WEB3 PROVIDER.

In order to deploy to a different network, the truffle config file can be modified to include the host, port and id of the network where the contracts are to be deployed.

To test if the deployed solution is working, we can buy an item from the shop, use the link provided through email to rate a seller. Click yes to provide a positive score to the seller and if a modal shows up to confirm that the score has been added, then the application is working as intended. This is because adding a score to the blockchain requires communication between the data service smart contract and the oracle in order to encrypt the provided score and add it to the smart contract which will emit the event of ScoreAdded. The front-end web interface will only trigger the modal when the event is emitted (see fig. 15).

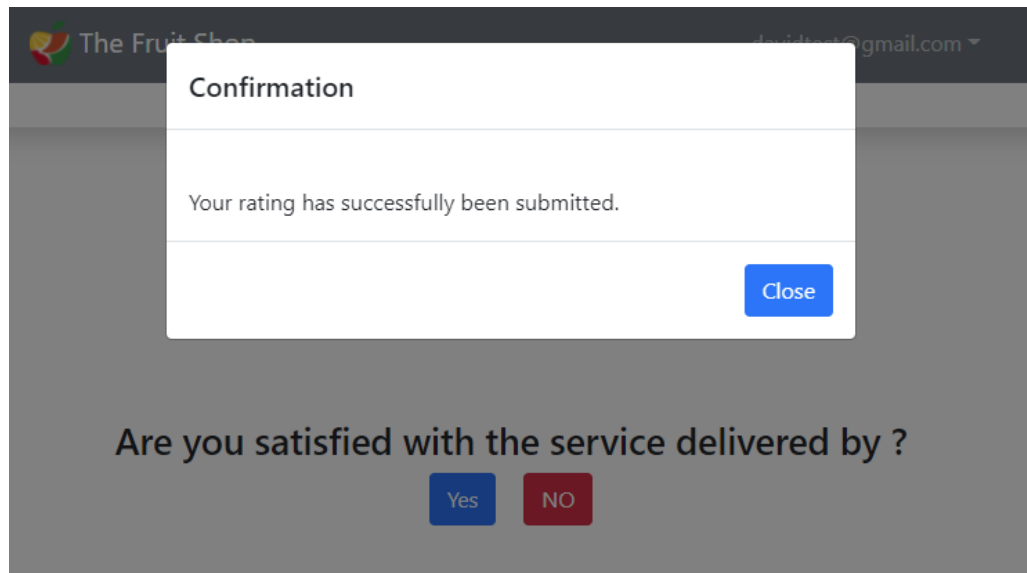


FIGURE 115 - MODAL CONFIRMATION TO NOTIFY THAT THE SCORE HAS SUCCESSFULLY BEEN ADDED TO THE BLOCKCHAIN.

---

## 1.7 HOW TO TEST

---

As our software includes different components which have specific dependencies, we describe them individually in the interest of readers' understanding.

---

### 1.7.1 Tests for Web Interfaces

---

Function	Input	Expected Output	Actual Output
Launch Application	Npm start	Opens the Fruit shop application	As expected
Adding items to the cart	The user clicks on "add to cart" button.	The item will be added to the cart.	As expected
Removing items from the cart	The user clicks on remove Item button on cart.	The item will be removed from the cart.	As expected
Buying items	The user will click the "buy now" button	Email is sent with confirmations of the items with links to submit feedback of the items bought.	As expected
Accessing the feedback page with the links provided from Gmail.	The users click on the feedback link.	New tab is opened displayed with a question	As expected
Buyer submits the feedback	Buyer clicks on their preferred answer.	The user is displayed with notification that their feedback is successfully collected.	As expected
Users accessing dashboard	http://localhost:3000 /dashboard	Displays the dashboard	As expected
Multiple options available to query	Click on choose query	Displays 2 query options	As expected
Check Individual user feedback for user in the system	Input 120 for user id	Display individual score of the user ID	As Expected
Check Individual user feedback for user not in the system	Input 1992 for user id	Display error-unable to find user id	As Expected

The table shows some of the UI related functional tests that were carried out.

### 1.7.2 Tests for Aggregator Module

Several tests have been performed to ensure that the application performs as expected.

These tests range from checking that an event gets emitted when a score is added, querying user ratings on the dashboard to testing the endpoints of the swagger API.

Function	Input	Output	Coverage
Description	Description	Description	Description
Testing that when a score gets added to the blockchain, an event called ScoreAdded gets emitted score	Provide a score to a seller using the link sent through email.	Modal triggered when the event is emitted to alert the user that the score has been added to the blockchain	Coverage satisfactory
Testing that aggregation happens when the 5th score for a seller is submitted. Four scores have been submitted for the specific seller.	Providing a rating to a seller who has been rated by four buyers.	After the fifth seller has been submitted, the score for the seller is automatically aggregated and added to the blockchain. A transaction hash and timestamp is added to the off-chain storage.	Coverage satisfactory
Verify that a used token cannot be re-used for the same seller	Use a token that has already been used to rate a seller.	A modal alerts the user that the token is not valid	Coverage satisfactory
Dashboard returns the individual score of a buyer that already rated a seller	Provided a user id, the corresponding individual score would be the output	The encrypted individual score of the user	Coverage passable. Dashboard returns the first encountered encrypted score that

			matches the user id.
<b>Get_reputation</b> endpoint returns the decrypted aggregated score of a seller	Seller ID	The decrypted aggregated score of the seller	Coverage satisfactory
<b>Post reputation_score</b> to aggregate seller score using individual scores of users who rated the seller.	Seller ID	Computes the aggregated score using the individual scores of users who rated the seller. The aggregated score is added to the blockchain and also returned.	Coverage satisfactory
<b>Verify reputation</b> to return the decrypted aggregated score of a seller, the transaction hash and the timestamp	The seller id whose score is being verified	Outputs decrypted aggregated score, transaction hash and timestamp	Coverage satisfactory
<b>Individual score</b> returns the individual score of a user who rated a specific seller	Seller ID and user ID	Encrypted individual score of the user who rated the seller	Coverage satisfactory
<b>Individual_scores</b> returns an array of individual scores who rated a seller	Seller ID	Returns all the encrypted individual scores of users who rated the seller	Coverage satisfactory
<b>Token_used</b> tells us whether a token was used to rate a seller	Seller ID and token	Returns a Boolean to inform whether or not the token has been used.	Coverage satisfactory

## 2 TECHNICAL VALUE ADDED

**Hint:** the technical performance includes free form paragraphs 2.1,2.2, 2.3 and table form paragraph 2.4

### 2.1 KEY INNOVATIONS OF YOUR SOLUTION, IN SUMMARY

This project makes following innovative contributions

1. A blockchain based decentralized reputation system for online marketplaces where the sellers and buyers submit their feedback without being worried about their private feedback scores or personal information. This helps marketplace to limit the effect of retaliation against each other as feedback scores are in encrypted form.
2. The system performs its function in a decentralized way as no centralized entity is involved in the computation process and user's feedback are only revealed in the aggregate form.
3. Provenance-aware verifiable reputation modelling such that user or marketplace sellers can verify reputation through off chain data in a decentralized way.
4. And finally, End-to-end decentralisation Interoperability with other services of the ecosystem.
5. Due to the distributed and decentralized characteristics of proposed blockchain structure and embedded mechanism within the approach, it would be very difficult for attackers to tamper the reputation data or feedback score stored on the blockchain (off chain or on chain).

### 2.2 EVENTUAL SUGGESTED EVOLUTIONS OF YOUR SOLUTION

The following are some of the potential evolutions to REPUTABLE:

1. To incorporate the mechanism to handle the unfair ratings this could be achieved through non-interactive zero knowledge proof of trustworthiness of scores and trustworthiness of score providers.

2. Another evolution could be incorporating mechanism to have a decentralized campaign mechanism.
3. The project can also be evolved by allowing user to provide comments in the privacy-preserving way.
4. Another evolution could be incorporating an incentive mechanism in the system so that if the users are providing feedback, they can get reward in the form of some discount or some monetary value and this discount or monetary value increase or decrease with the honesty of the feedback. This is very challenging task but can be incorporated in the set with small independent changes in the approach.

## 2.3 PRESENT OR FUTURE PATENTABILITY OF YOUR SOLUTION

We have done preliminary research with regards to patentability of the REPUTABLE solution. In the present form the solution is not patentable but with some more contribution and results, the solution may be patentable. This because of the fact that project applies off-chain and on-chain blockchain setup for verifiability and processing of user feedback score and reputation values.

## 2.4 TECHNICAL KPIS

**Hint:** the following tables summarize the technical KPIs of your solution. It is suggested that you compile first 2.1, 2.2 and 2.3 before compiling KPI, in standard format. Since some of KPIs can't apply to call 1 or apply only to some projects: ask your coach if some KPIs does not apply to your project



#### 2.4.1 INTEROPERABILITY AND STANDARDIZATION

KPI	Can apply to your project?	Did you implement?	Summarize your contribution
Interoperability with other components of ONTOCHAIN	Yes	Yes	Through aggregator API

#### 2.4.2 KPIS TOWARDS INNOVATION

KPI	Can apply to your project?	Did you implement?	Summarize your contribution
Did you implemented new innovative ONTOCHAIN use cases?	No	No	Not applicable
Did you implement new innovative ONTOCHAIN reasoning technologies?	No	No	
Did you implement new ways to serialise ontologies and semantics on blockchain?	No	No	
Did you implement other new approaches to implementing "immutable" semantics and reasoning?	No	No	

#### 2.4.3 KPIS TOWARDS MORE HUMAN-CENTRIC EVOLUTION OF THE INTERNET

KPI	Can apply to your project?	Did you implement?	Summarize your contribution
Which is the Trust Assessment Effectiveness, e.g., accuracy for subjects or for content, for your solution?	Yes	Yes	Please refer to Appendix A for details.

How can you assess the privacy/anonymity of your solution?	Yes	Yes	Please refer to Appendix A for details
Did you implement of zero knowledge proof protocols?	Yes	Partially	Implemented in isolation but not integrated with the final product
Details the size of decentralized storage, in GB	Yes	Yes	Less than a GB for current version but will depend on the use and integration with the volume of applications
Detail the number of applications deployed.	Yes	Yes	1 (a dummy marketplace created)

#### 2.4.4 KPIS TOWARDS MORE DECENTRALIZED NGI

KPI	Can apply to your project?	Did you implement?	Summarize your contribution
Did you implemented new decentralised computing technologies for storing and accessing data, e.g., via the OAI- PMH protocol, that achieve high reliability, availability, Quality of Service, and similar properties necessary to realise new decentralised services?	Yes	Partially	<p><b>Data Storage:</b> our solution benefits from decentralised off-chain storage to store individual feedback. Currently, we use cloud storage for this feature however this can be extended to services such as filecoin etc.</p> <p><b>Data Processing:</b> We developed a proprietary oracle to achieve aggregation of individual feedbacks. Our oracle is currently decentralised and can benefit from decentralisation</p>

Did you implement new decentralised social networks?	No	No	
Did you implement new decentralised publishing platforms?	No	No	
Did you implemented new Digital Twin technologies that can help establish digital representation of the reality in specific circumstances where needed?	No	No	

#### 2.4.5 KPIS TOWARDS NEW FORMS OF INTERACTION AND IMMERSIVE ENVIRONMENTS FOR NGI USERS

KPI	Can apply to your project?	Did you implement?	Summarize your contribution
Did you implemented new human to Internet interaction paradigms developed in the use cases of ONTOCHAIN?	No	No	Text (30 words or less)
Did you implemented decentralized apps in ONTOCHAIN that involve human interactions in education, energy, finance, governance, healthcare, identity, interoperability, mobility, privacy, public sector, real estate, social impact, supply chain?	No	No	

#### 2.4.6 KPIS RELATED TO THE IMPLEMENTATION

KPI	Can apply to your project?	Did you implement?	Summarize your contribution
Code simplicity (analyser used and results)	Yes	Yes	Code includes comments and details about dependencies have been provided
Testability Coverage (method/tool used a results)	Yes	Yes	Detailed guidance is included

### 3 COMMUNICATION, DISSEMINATION, EXPLOITATION

#### 3.1 YOUR CONTENTS

##### 3.1.1 MAIN ACHIEVEMENTS

ONTOCHAIN aims to develop a trustworthy platform for content sharing and information exchange utilising cutting-edge within blockchains, reputation systems, and semantic web technologies. Establishing trustworthy information exchange becomes a challenge which requires mechanisms to determine trustworthiness of external services. Through the work conducted within this project, we identified the fundamental requirements for an effective reputation system for ONTOCHAIN. These include:

- User-centric reputation modelling and calculation
- Privacy-preserving user engagement
- Provenance-aware verifiable reputation modelling
- End-to-end decentralisation
- Interoperability with other services of the ecosystem

REPUTABLE addresses these requirements and has the potential to deliver a cross-platform privacy-aware reputation system which leverages blockchain technology to achieve decentralised, verifiable calculation of reputation scores. Further it enables interaction with end users and systems through a secure, reputation analytics dashboard to facilitate user verification as seamless integration with other systems and services.

##### 3.1.2 DEMO

(Develop a 5-minute demo video of the solution developed. Start by describing the challenge addressed, your approach, your achievements, and key benefits for the ONTOCHAIN ecosystem and end users. The video should be recorded in a horizontal format and have a good resolution) Example:

<https://www.dropbox.com/s/94kj68kwa8o3wx4/Reputable%20demo.mp4?dl=0>

In this document you have only to provide the link to the video you developed and a sentence to summarize your video.

### 3.1.3 YOUR TESTIMONIAL

Collaborative R&D is the ethos of the ONTOCHAIN initiative and it has also been our primary motive to join the programme. Through this programme, we have been able to advance our ideas in a structured way to a proof of concept implementation. The governance structure adopted by the programme has been really useful in having detailed technical discussions on a weekly basis which has also ensured timely delivery of our objectives and KPIs. **[Dr Junaid Arshad]**

## 3.2 COMMUNICATION, DISSEMINATION, EXPLOITATION RESULTS

### 3.2.1 NEWS

- An article disseminating REPUTABLE project and its value to blockchain-based ecosystems, BCU web, August 2021. (ref: <https://www.bcu.ac.uk/computing/research/cyber-physical-systems/research-projects/reputable>)

### 3.2.2 PRESS RELEASES

- An article disseminating REPUTABLE project and its value to blockchain-based ecosystems, BCU web, August 2021. (ref: <https://www.bcu.ac.uk/computing/research/cyber-physical-systems/research-projects/reputable>)

### 3.2.3 SCIENTIFIC PAPERS

We are in process of finalising our paper based on REPUTABLE which we intend to submit to the ACM Journal of Distributed Ledger Technologies (<https://dl.acm.org/journal/dlt>).

### 3.2.4 EXHIBITIONS

- Presentation of REPUTABLE concept design at the Security and Trustworthy Systems research cluster (BCU) meeting - May 2021

- Research presentation of the REPUTABLE core concept at the University of Derby, June 2021.

---

## 4 APPENDIX: FINAL DESIGN UPDATES

---

(In the eventual case that your final implementation has some differences from the design you have already described in the deliverable D3, please document these differences in this appendix).

If the design of your final implementation is completely described by the deliverable D3, simply write: "The design of final implementation is completely described by the deliverable D3".

We summarise differences between the architecture presented in D3 and our implementation below.

- We planned to use an existing decentralised oracle service however we encountered technical challenges making existing services unsuitable for our implementation. Therefore, we developed a proprietary oracle to fulfil our objectives however this oracle is currently not centralised
- We planned to store user feedback directly to a data service however while investigating implementation strategies such as side chains, we have implemented the user feedback storage directly to the chain via a smart contract. Although this is different to our original plan, we believe this is an important step for future development of the REPUTABLE system (such as to exploit use of sidechains to store user feedback and aggregate score).
- Our plan was to use zero knowledge proofs to ensure privacy of user feedbacks and although we implemented these however integration of these within the final product was extremely challenging due to the support for such advanced operations within solidity language. Therefore, we regard integration of zero knowledge proofs with the current implementation an important future step.



## APPENDIX A

### **Privacy-aware reputation system - Verifiability of output**

In this scheme, the aggregated score of the retailer or seller is publicly available on the bulletin board along with non-interactive zero-knowledge proofs. The proofs provide assurance that the provided encrypted ratings are well formed. Further, NIZK proofs provide public verifiability for the computation of the aggregated score. If a retailer displays the aggregated score on their website then everyone can check its correctness by looking into the bulletin board. As such, no retailer can misrepresent their own aggregated score on their web page without being caught.

### **Privacy-aware reputation system - Unlink-ability**

In this section, we show that our scheme protects privacy of the participants of the protocol. Our main result is in Lemma 1. This lemma proves that if there are at least a pair of participants, who provided feedback differently (i.e. one of them submits 0 as an input and the other one provides 1 as an input) then the adversary will not be able to breach the privacy of any of the two participants. In other words, if the feedback values of all the honest participants are not the same, then the adversary will not be able to learn the feedbacks of any of the honest participant. Note that, if the aggregate score of all the participants (both honest and colluding) is known by the end of the protocol operations and the adversary knows the inputs of the colluding users, then the adversary can trivially learn the aggregate or sum of all the inputs of honest users. Now, if all the inputs of the honest users are the same, then the sum will simply expose the inputs of all the honest users. This is because, if all the honest users submit 0 as inputs, then the sum of their inputs will be 0, and if they all submit 1 as inputs, then the sum of their inputs will be equal to the number of honest users. So, in order to preserve the privacy of all the honest users, we must have at least a pair of honest users, who provided distinct inputs. More precisely, the adversary learns nothing more than the aggregate of all the inputs of honest users. Lemma 1 proves this fact. We prove Lemma 1 by reducing it to the well-known Decisional Diffie Hellman problem (DDH). Hence, if the DDH problem is intractable in the mathematical group  $G$ , the REPUTABLE protocol is secure.

**Assumption 1. DDH assumption:** Given  $g, g^a, g^b$ , and  $\Omega \in \{g^{ab}, g^{ab}g\}$ , it is hard to decide whether  $\Omega = g^{ab}$  or  $\Omega = g^{ab}g$ .

**Lemma 1.** If there exist at least two participants  $U_{\tau_\alpha}$  and  $U_{\tau_\beta}, \alpha, \beta \in [\eta], \alpha \neq \beta$ , such that  $v_{\tau_\alpha r} + v_{\tau_\beta r} = 1$  for some



seller  $r \in [n]$ , then no adversary can distinguish between following two cases:

- 1  $v_{\tau_{\alpha}r} = 1, v_{\tau_{\beta}r} = 0$
- 2  $v_{\tau_{\alpha}r} = 0, v_{\tau_{\beta}r} = 1$

Proof. We show that an adversary  $\mathcal{A}$  who can distinguish between two bulletin boards where the values of  $v_{\tau_{\alpha}r}$  and  $v_{\tau_{\beta}r}$  are interchanged, then  $\mathcal{A}$  could be used to construct another adversary  $\mathcal{B}$  against the assumption 1.  $\mathcal{B}$  works as follows:

it receives as input  $g^a, g^b$  and a challenge  $\Omega \in \{g^{ab}, g^{ab}g\}$ .

It allows  $\mathcal{A}$  to choose a set of secret keys  $\{x_{\tau_{i}r} : i \in [\eta] \setminus \{\alpha, \beta\}\}$ .  $\mathcal{A}$  also chooses the set of scores  $\{v_{\tau_{i}r} : i \in [\eta] \setminus \{\alpha, \beta\}\}$ . Then  $\mathcal{A}$  sets  $X_{\tau_{i}r} = g^{x_{\tau_{i}r}}$ .  $\mathcal{B}$  sets  $X_{\tau_{\alpha}r} =$

$g^{x_{\tau_{\alpha}r}} = g^a$  and  $X_{\tau_{\beta}r} = g^{x_{\tau_{\beta}r}} = g^b$ . Now,  $\mathcal{A}$  computes the

set of ballots for each participant  $U_{\tau_i} : i \in [\eta] \setminus \{\alpha, \beta\}$  as follows:

$c_{\tau_{i}r} = g^{x_{\tau_i} r y_{\tau_i}} g^{v_{\tau_i} r} : i \in [n] \setminus \{\alpha, \beta\}$ .  $\mathcal{B}$  sets  $c_{\tau_{\alpha}r} = (g^a)^{z_{\tau_{\alpha}r}} g / \Omega$  and  $c_{\tau_{\beta}r} = (g^b)^{z_{\tau_{\beta}r}} * \Omega$ .

Here,  $z_{\tau_{\alpha}r} = \sum_{i=1}^{\alpha-1} x_{\tau_i r} -$

$\sum_{i=\alpha+1}^{\beta-1} x_{\tau_i r} - \sum_{i=\beta+1}^{\eta} x_{\tau_i r}$  and  $z_{\tau_{\beta}r} = \sum_{i=1}^{\alpha-1} x_{\tau_i r} + \sum_{i=\alpha+1}^{\beta-1} x_{\tau_i r} - \sum_{i=\beta+1}^{\eta} x_{\tau_i r}$ .  $z_{\tau_{\alpha}r}$  and  $z_{\tau_{\beta}r}$  can

be computed by  $\mathcal{B}$  with the help of  $\mathcal{A}$ , as the values of  $x_{\tau_i r}$  are chosen

by  $\mathcal{A}$  for all  $i \in [n] \setminus \{\alpha, \beta\}$ . Now note that, if  $\Omega = g^{ab}$ , then  $c_{\tau_{\alpha}r} = R_{\tau_{\alpha}r}^{x_{\tau_{\alpha}r}} g$  and

$c_{\tau_{\beta}r} = R_{\tau_{\beta}r}^{x_{\tau_{\beta}r}}$ . That is

if  $\Omega = g^{ab}$ , then  $v_{\tau_{\alpha}r} = 1$  and  $v_{\tau_{\beta}r} = 0$ . Alternatively, if  $\Omega = g^{ab}g$ , then  $c_{\tau_{\alpha}r} = R_{\tau_{\alpha}r}^{x_{\tau_{\alpha}r}}$  and  $c_{\tau_{\beta}r} = R_{\tau_{\beta}r}^{x_{\tau_{\beta}r}} g$ . That is if

$\Omega = g^{ab}g$ , then  $v_{\tau_{\alpha}r} = 0$  and  $v_{\tau_{\beta}r} = 1$ . If  $\mathcal{A}$  can distinguish between these two cases,  $\mathcal{B}$  can identify  $\Omega \in \{g^{ab}, g^{ab}g\}$  correctly.

### Privacy-aware reputation system - Correctness of Protocol

**Lemma 2.** In the existing REPUTABLE system, participants can learn the correct aggregated reputation of other participants. We prove this under the model where participants are honest in providing their feedback, but they try to learn the feedback score of other participants.

Proof. For any seller  $r$ , feedbacks provided by the participant  $U_{\tau_{i}r}$  is  $c_{\tau_{i}r} = g^{x_{\tau_i} r y_{\tau_i}} g^{v_{\tau_i} r}$ . The product of all cryptograms is given by  $\theta_r =$

$$\prod_{i=1}^{\eta} c_{\tau_{i}r} = \prod_{i=1}^{\eta} g^{x_{\tau_i} r y_{\tau_i}} g^{v_{\tau_i} r} =$$

$g^{\sum_{i=1}^{\eta} x_{\tau_i} r y_{\tau_i} + \sum_{i=1}^{\eta} v_{\tau_i} r}$ . We know that  $\sum_{i=1}^{\eta} x_{\tau_i} r y_{\tau_i} = 0$ , then  $\theta_r = g^{\sum_{i=1}^{\eta} v_{\tau_i} r}$ , or

$\sum_{i=1}^{\eta} v_{\tau_{ir}} = \log_{\theta_r}$ . This proves computation correctness of the protocol.

### Attacks on the System

The scheme is secure as long as for each  $r \in [n]$ , there are at least two participants  $P_{\alpha}$  and  $P_{\beta}$  satisfying  $v_{\alpha r} + v_{\beta r} = 1$ , that is if  $P_{\alpha}$  and  $P_{\beta}$  assigns different scores to  $P_r$ . Let us suppose, the attacker has colluded with some  $c$  participants to deduce the score  $v_{\alpha r}$  assigned to  $P_r$  by an honest participant  $P_{\alpha}$ . Alternatively, the attacker can also launch a Sybil attack by creating  $c$  fake participants to deduce the score  $v_{\alpha r}$ . Now, if there exists at least one uncompromised participant  $P_{\beta}$  such that either  $v_{\alpha r} = 0 \wedge v_{\beta r} = 1$  or  $v_{\alpha r} = 1 \wedge v_{\beta r} = 0$ , then it would be computationally infeasible for the attacker to find  $v_{\alpha r} = 0$  or  $v_{\beta r} = 1$ . The scheme assumes that all the participants who completed step 2 of the reputation aggregation protocol would provide their feedback. If some participants abort, the system cannot compute the aggregated reputation in that iteration. As such, the iteration has to be started afresh. Though, this kind of denial of service attacks cannot compromise the privacy of any participant, an attacker can use this as a means to disrupt the normal flow of the protocol. A participant who has intentionally aborted the protocol in the middle should be excluded from the network in order to ward off possibilities of sabotage in the future.

### Privacy-aware reputation system - Privacy and integrity analysis

In this section, we analyze privacy aspects of participants in the reputation aggregation process. At the end of the reputation aggregation process, each participant or the marketplace can only hold the global reputation score of a particular retailer, which cannot be linked to infer the individual feedback of users, neither can be used to infer who voted positively or negatively for the retailer. The published feedback is the valid score of either 0 or 1 in the following format  $g^{xy}g^v$  for  $v = 0$  or 1. It is indistinguishable from random feedback and the associated 1-out-of-2 ZKP reveals nothing more than the statement: the  $v$  is either 0 or 1. The encrypted feedback value and computation on the encrypted data ensure that participants would not learn anything about the feedback value other than the final aggregated reputation score. The scheme is also secure if a number of feedback providers collude with each other to learn the feedback score of some target user. However, the scheme can reveal the feedback value when  $n - 1$  participants collude against the single remaining user. Note that, the final aggregate of all feedbacks has to be made public. Hence, it is impossible to ensure the privacy of an honest feedback provider when all other feedback providers collude against her. The

protocol assures the maximum possible privacy for any honest feedback providers, which a scheme of this sort can achieve. Moreover, our protocol does not require any centralized trusted third party for the generation of cryptographic parameters and can provide reputation aggregation even if a certain participant is not online at the time of the reputation aggregation.

### **Privacy-aware reputation system - Computation Complexity**

We present the computation costs in terms of time require for creating the cryptograms (the encrypted feedback and the non-interactive zero-knowledge proof ) at the client side, and the time require for aggregating the cryptograms from the bulletin board. Table 1 presents a comparison of the existing REPUTABLE system architecture and computational complexity with other reputation systems. The computation and communication complexities of the privacy-preserving reputation system for the client and the analyst are given in Table 2. Table 3 presents the computation time and communication overhead required for verifying the reputation statistics.

### **Evaluation Metrics:**

In addition to the theoretical analysis above, we have conducted empirical evaluation to measure the efficiency of the REPUTABLE system. We share specific measurements below.

Measurement type	Measurement value	Rationale
The size of the individual user feedback	128 bytes	Memory and storage requirements
The size of the aggregate score	128 bytes	Memory and storage requirements
The time taken by the 'Data Service SC) to store the individual feedback to blockchain.	15.62ms	Time analysis
The time taken by the 'aggregator' to create the aggregate score	1684.5ms	Time analysis

The time taken by the 'oracle' to store aggregate score on the chain.	421.125ms	Time analysis
The time taken by the aggregator API to <i>GET</i> reputation score	117.30ms	Time analysis
The time taken by the aggregator API to <i>POST</i> reputation score	1080ms	Time analysis
The time taken by the aggregator API to verify reputation score	735.70ms	Time analysis
The time taken by the aggregator API to <i>GET</i> individual score for a seller	103.89ms	Time analysis
The time taken by the aggregator API to <i>GET</i> individual scores for a seller	434.36ms	Time analysis
The time taken by the aggregator API to verify token used	158.39ms	Time analysis
The time taken by dashboard to query aggregate score for a seller	3582.9ms	Time analysis
The time taken by dashboard to verify aggregate score for a seller.	0.4ms	Time analysis
The time taken by the dashboard to query the individual feedback for a user	0.3ms	Time analysis
Time taken to generate keys	62.5ms	Time analysis
Time taken to generate the token	0.2ms	Time analysis