IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Trustworthy Blockchain Gateways for Resource-Constrained Clients and IoT Devices

MAZIN DEBE[1], KHALED SALAH[1], RAJA JAYARAMAN[2], JUNAID ARSHAD[3]
[1]Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology, Abu Dhabi, UAE
[2]Department of Industrial and Systems Engineering, Khalifa University of Science and Technology, Abu Dhabi, UAE
[3]School of Computing and Digital Technology, Birmingham City University, Birmingham, UK

Corresponding author: Mazin Debe (e-mail: mazin.debe@ku.ac.ae).

ABSTRACT

Constrained blockchain clients are unable to process and store the entire blockchain ledger and mine blocks on the blockchain. Such nodes rely on the view of blockchain provided by full nodes termed as *gateways*. However, gateway nodes can provide a distorted view of the blockchain, making lightweight clients vulnerable to eclipse attack. When under such an attack, a client cannot differentiate between a forked view of the blockchain and the legitimate blockchain ledger leading to fatal consequences and huge losses incurred. To mitigate such threats, we propose a data attestation solution which employs full nodes as validators to attest the responses reported by gateways of lightweight nodes. Leveraging smart contracts, our approach gives lightweight clients confidence in the data reported as they are unable to validate it from the blockchain network itself. The system governs the attestation process that comprises of submitting attestation requests, approving them, recording the response of validators, and manage payments. Clients can, thereafter, provide their feedback about the validator/gateway performance in the form of a reputation score. We present the proposed system architecture and describe its implementation on the Ethereum blockchain network. We evaluated the proposed solution with respect to functionality testing, cost of execution, and security analysis of smart contracts developed. We have also made our smart contracts code publicly available on Github. [1].

INDEX TERMS Blockchain, Ethereum, IoT, Gateways, Trust, Lightweight nodes, Full nodes

I. INTRODUCTION

With the increasing adoption of blockchain technology has resulted in growing use of lightweight clients to support integration between blockchain and the Internet of Things (IoT). The integration of these technologies has facilitated development of novel applications as well as strengthening existing use-cases. These include applications in Artificial Intelligence, reputation systems, IoT monetization, asset management, as well as in the shipping industry [1]–[4].

A primary motivation in using IoT devices is their ability to provide lightweight, connected solutions to diverse domains. However, such devices are typically constrained with respect to their abilities in computation and memory [5], [6]. These restrictions are particularly problematic when utilising the blockchain technology as a backend to serve such clients. Blockchain consensus protocols typically require a client (full node) to download the entire blockchain ledger to achieve synchronization among the participants of the network. The requirements of a full node are evidently not attainable by all clients due to the large storage space and processing power needed. Most clients, such as mobile wallets and IoT devices, often opt to use Simplified Payment Verification (SPV) to check the integrity of transactions and blocks, without the need to download the complete blockchain [7]. Such clients, or lightweight nodes, rely on a full node to fetch data from the blockchain.

Lightweight nodes send and receive data and make decisions based on the copy of the blockchain provided by the reference full node. This insinuates that the full node is a gateway for all of the connected light nodes to the actual blockchain.

[1]https://github.com/MazenDB/Gateways/

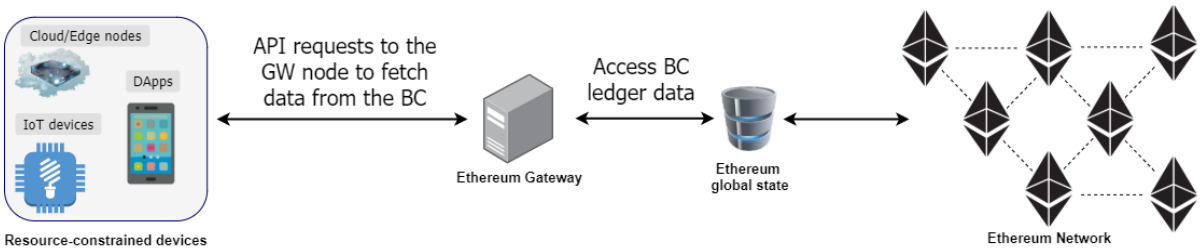

FIGURE 1: Typical connection between lightweight blockchain clients and the blockchain network through a gateway, making clients vulnerable to cyber attacks

Naturally, light nodes need to trust this gateway in order to connect to it and perform any transactions. This contradicts the intrinsic trustless characteristics of the blockchain. These gateways can single-handedly reject a transaction without the consensus of the rest of the blockchain network. Although light clients can utilize the SPV method to verify transactions using block headers, it does not provide enough information to make an informed decision. Using malicious gateways could result in adverse consequences in terms of data leaks, wrongfully addressed cryptocurrency transfers, and biased business decisions. The phenomenon where a gateway provides a falsely forked view of the blockchain to its clients is called an eclipse attack [8].

A typical connection between a lightweight node and the blockchain network is shown in Fig.1. Mobile DApps, IoT devices, and other lightweight clients rely completely on the copy of the blockchain provided by the gateway. Therefore, these devices have no way of validating the information provided by the referenced full node. Therefore, a mechanism for validation of data provided by the gateway is required. Such mechanism will enable clients to be confident in the data and make decisions and transactions, respectively. In this paper, a data attestation approach is presented which utilises available gateways in the blockchain network as validators for the information provided by other gateways. Clients can send data that require attestation to the validators, which then validate the request and return their response to the light client. This mechanism includes multiple aspects such as the validator selection mechanism, recording responses from the validators, managing payment, and rating gateways. All of these are governed by Ethereum smart contracts, as these record all interactions on a permanent distributed ledger.

The major contributions of this paper are:

- We propose a data attestation approach that leverages Ethereum smart contracts to govern validation of the requests from Ethereum lightweight clients. This solution accepts attestation requests from light nodes and assigns respective validators to verify the information reported by the gateway.
- We present detailed description of the different components of the proposed approach, including the selection mechanism, attestation recording, and reward transfers. We also propose a way to rate gateways and solve potential disputes between clients and validators.
- We provide implementation details via Ethereum smart contracts for the proposed system. In addition, we show the series of transactions for all the presented functionalities. Deployment of the smart contracts is simulated with the aid of the Remix IDE (blockchain-based development environment).
- We evaluate the proposed data attestation solution by validating its different functionalities, assessing performance, estimating execution costs, and discussing the security aspect of the system by showcasing several possible scenarios. For further analysis, we present a comparative analysis of the proposed method with relevant existing solutions.

The remainder of this paper is organized as follows: Section II discusses background information about the blockchain technology and the eclipse attack. Section III briefly discusses several efforts in the literature that is in the same domain of this paper. Section IV presents the design and details of the proposed approach. Section V highlights the implementation details. Section VI discusses the evaluation of our solution, which includes functionality testing and cost analysis, in addition to some of the challenges faced and possible future improvements to this work. We present our conclusions in section VII.

II. BACKGROUND INFORMATION

This section explains some background information about the blockchain technology and the eclipse attack to provide important context to the rest of the paper.

A. BLOCKCHAIN TECHNOLOGY

A blockchain network, such as Bitcoin and Ethereum, consists of a series of blocks that are linked together to form a tamper-proof ledger [7]. Each block holds several transactions representing interactions between different participants of the network. The contents of a block are hashed and included in the following block to form the said chain. Hashing this information, as well as grouping transactions to form a block, is done by special nodes called miners. Miners run specific code (consensus algorithm) to be able to append blocks and evidently transaction to the existing chain.

Regular blockchain clients are not required to run consensus algorithms and are only required to submit transactions, and in some blockchain networks, pay a fee for the transaction. Numerous consensus algorithms are implemented to determine the rules of appending new blocks to the chain. Each application of the blockchain implements an approach that suits its use case. Some popular examples of consensus algorithms include Proof of Work (PoW), Proof of Stake (PoS), and Practical byzantine fault tolerance (PBFT).

The entire chain of blocks can be downloaded and stored on any of the devices participating in the blockchain network. That is why the blockchain network is viewed as a distributed ledger and can be considered a decentralized system. In some cases, the blockchain network is open to the public, such as in the Bitcoin and Ethereum networks. This is called a public blockchain, in contrast to private blockchains that provide access only to a restricted group of people. Both types of blockchain networks are permanent and tamper-proof. Any transaction that is added to the network cannot be removed later as it jeopardizes the integrity of the ledger. Another difference between various blockchain networks is the ability to modify the business logic of the blockchain. Ethereum, for instance, supports customized rules to achieve this through smart contracts however other blockchain platforms such as Multichain do not support smart contracts.

B. ECLIPSE ATTACK

Blockchain miners and clients can download the entire data on the blockchain from other peers to validate previous as well as new transactions. Members of the network that do so are referred to as full nodes and are normally either miners or service providers. In contrast, light nodes that cannot afford the storage and processing power requirements of a full node use a copy of the blockchain extracted from a full node. This is not consistent with the decentralization of the blockchain network, as an attacker can exploit lightweight clients as it controls its communication with the blockchain network as discussed in [9]. This attack is known as an eclipse attack and is discussed in detail by Heilman *et al.*as they explore its devastating effects on the Bitcoin network. In an eclipse attack, the victim is isolated from the real network activity and given a distorted view instead. In fact, it is easier for the attacker to target lightweight clients as they need to hijack significantly fewer connections between the client and the blockchain network. This results in financial losses such as double spending and ill-informed business decisions. Eclipse attacks are executed to infiltrate the target device or to attack the blockchain network itself.

III. RELATED WORK

In this section, we will discuss some of the solutions in the literature aimed to address the threat of eclipse attacks on lightweight clients connecting to gateways that provides them access to the blockchain.

Alangot *et al.*discussed two approaches to detect eclipse attacks on lightweight Bitcoin clients [8]. The first approach assumes that attackers need significantly more time to create false blocks. This is due to the fact that one entity does not have enough resources to form blocks as fast as the rest of the network. Therefore, an unusual spike in creation times insinuates an attack. The second proposed solution is a gossip protocol. In this approach, a client obtains its blockchain view from a server that provides its strongest view based on input from all clients. These approaches modify the blockchain infrastructure and are implemented on top of the existing protocols. The proposed modifications are tailored to help prevent specifically eclipse attacks on light blockchain clients. However, the gossip protocol is computationally expensive for light clients with minimal processing power.

Kiayias *et al.*propose an improvement to the current Proof of Work (PoW) consensus protocol [10]. The novel Non-Interactive-Proofs-of-Proof-of-Work (NIPoPoWs) enhances the performance of blockchain networks based on PoW such as Bitcoin and Ethereum and expands their functionalities. It does so by further reducing the requirements for light blockchain clients in terms of resources and processing ability in contrast to the traditional Simplified Payment Verification (SPV). In SPV, light clients are required to download block headers instead of entire blocks due to their restricted storage space. Although this significantly reduces the required storage, which is suitable for mobile devices, for instance, it is still a significant amount of data (several Gigabytes for SPV client in Ethereum) to be stored in clients such as IoT devices. With the modified consensus algorithm, clients are only required to download a polylogarithmic number of block headers. However, the developers assume the block difficulty to be constant. This is untrue in the case of Bitcoin and Ethereum, as the block difficulty is not persistent and tends to change over time. Further, NIPoPoWs works under the assumption that the honest chain is not being altered by an attacker or a malicious entity.

The authors in [11] propose a transaction verification client called FlyClient which requires lightweight nodes to download only a logarithmic number of block headers. FlyClient creates shorter proofs as compared to NIPoPoWs using probabilistic sampling for blocks and Merkle Mountain Range (MMR) [12]. In addition, FlyClient causes minor changes to the current blockchain networks that it is applied to. The FlyClient approach requires that at least one honest miner be available at all times. This assumption is deemed to be too optimistic as IoT clients are susceptible to eclipse attacks, MITM, and other types of security attacks and compromises. Letz presents a novel approach in [13] comprising a superlight client. This approach helps prevent eclipse attacks by fetching data from a remote client while further reducing the demanded requirements by blockchain clients. While this approach overcomes the disadvantages of previous work, it is specifically designed for the Bitcoin network and not very favorable for blockchain networks with fast block confirmation, such as the Ethereum network.

IV. BLOCKCHAIN-BASED SOLUTION

We propose a data attestation solution leveraging Ethereum smart contracts which is able to validate the information provided by blockchain gateways to lightweight nodes as shown in Fig. 2. Light clients often connect to one gateway that they copy from their view of the blockchain. Although this gives devices with limited resources the opportunity to utilize blockchain technology, it compromises one of the biggest benefits of using the blockchain i.e. decentralization. Reliance on a single element to provide access to the entire network contradicts the consensus protocols that the blockchain network is based on. Our solution relies on using blockchain smart contracts that manage attestation of the data provided by the gateway. Therefore, the blockchain network chosen was Ethereum as it supports smart contracts and is a public blockchain that is accessible by all entities. Privacy of information is maintained nevertheless through restricted access to functions and data by users. All stakeholders are registered by the smart contract and are given an Ethereum Address as an identifier to track the process of attestation and solve disputes.

Light Ethereum clients contact full nodes to reference their copy of the blockchain. These clients request data from the gateway nodes through API requests. Full nodes are able to access the Ethereum global state ledger as they store the entire blockchain locally and append new blocks regularly as the chain of blocks is expanded. IoT devices, mobile clients, and other light nodes sometimes need to attest sensitive information that was reported by the gateway. They send this information to other full nodes that have full access to the blockchain. Clients gain more confidence in the data by consulting a higher number of full nodes. Full clients can serve as gateways for light clients as well data validators for clients connected to other gateways. Smart contracts manage the selection of validators to be fair for all available validators and maintain the quality of service. Light clients pay a fee for using this service respective to the number of attestations requested. On the other hand, validators get paid for validating this data as well.

Gateways/validators are assigned a reputation score demonstrate their trustworthiness. Therefore, they are incentivized to provide honest feedback, whether as gateways or validators, to avoid lowering their reputation score. Upon registration, they are assigned an average score which is modified based on their performance. After each interaction, clients provide feedback about the validators involved, which is translated to an increase or decrease in the reputation score of the validator. Validators are required to maintain a minimum score to be allowed to provide attestation services. Full nodes lose credibility by providing false information to clients, false attestation results, and failing to respond to attestation requests. In addition, validators are requested to deposit collateral as extra incentive to provide good service quality. Returning the deposit to a validator is contingent on them providing honest information and feedback to clients. Failing to do so results in deposit deduction and possible elimination from the list of validators.

Light clients submit a payment linear to the number of validators required along with the attestation request. The smart contract reviews the request and assigns validators in a round-robin fashion. The client is informed about the assigned validators in order to send them the requests securely and individually. Consequently, the validator informs the client about the result to be compared to that of the gateway. The smart contract is also informed of the result as well to maintain a record of the interaction. In addition, validators are paid their due amounts upon successful submission of their response. Throughout this process, all interactions related to a request is tracked via a request number generated by the smart contract at the request initiation level. This number maps to important information regarding the request including the requester address, number of requested validators, and current state of the attestation process. Records regarding all steps of the process are permanent as they are recorded on the immutable blockchain ledger and can be investigated later to solve disputes and disagreements. Due to the nature of the blockchain, function calls, events, and any alteration to the variables on the smart contract need to be stored in a block as a transaction and appended to the global blockchain.

Missing attestations can be reported as well via the smart contract through dedicated function calls. Clients typically would do so after a certain interval of time has passed since the request initiation. In addition, clients could report that they believe are misleading or dishonest. For instance, if one of the validators tries to claim that a request is invalid as opposed to the rest of the validators that approve it, the client can flag it as dishonest, which would have consequences on the validator by losing credibility and partial deduction of its deposit.

The process of attestation involves several stakeholders that interact together. The role of each one is explained further below.

- **Light clients:** These lightweight nodes include decentralized mobile applications, IoT devices, or any other resource-constrained device. Limited storage and processing power inhibits them from downloading the entire blockchain and making transactions independently. In order to access blockchain functionality by sending or receiving cryptocurrency and accessing smart contracts, they are required to refer to the blockchain view of some other full node. In our proposed solution, light clients can also request attestation from other full nodes to validate the information.
- **Validators:** Full nodes provide attestation services to light clients and get rewarded in *ethers* for that service. The performance of these validators is monitored by clients themselves that provide their evaluation. False

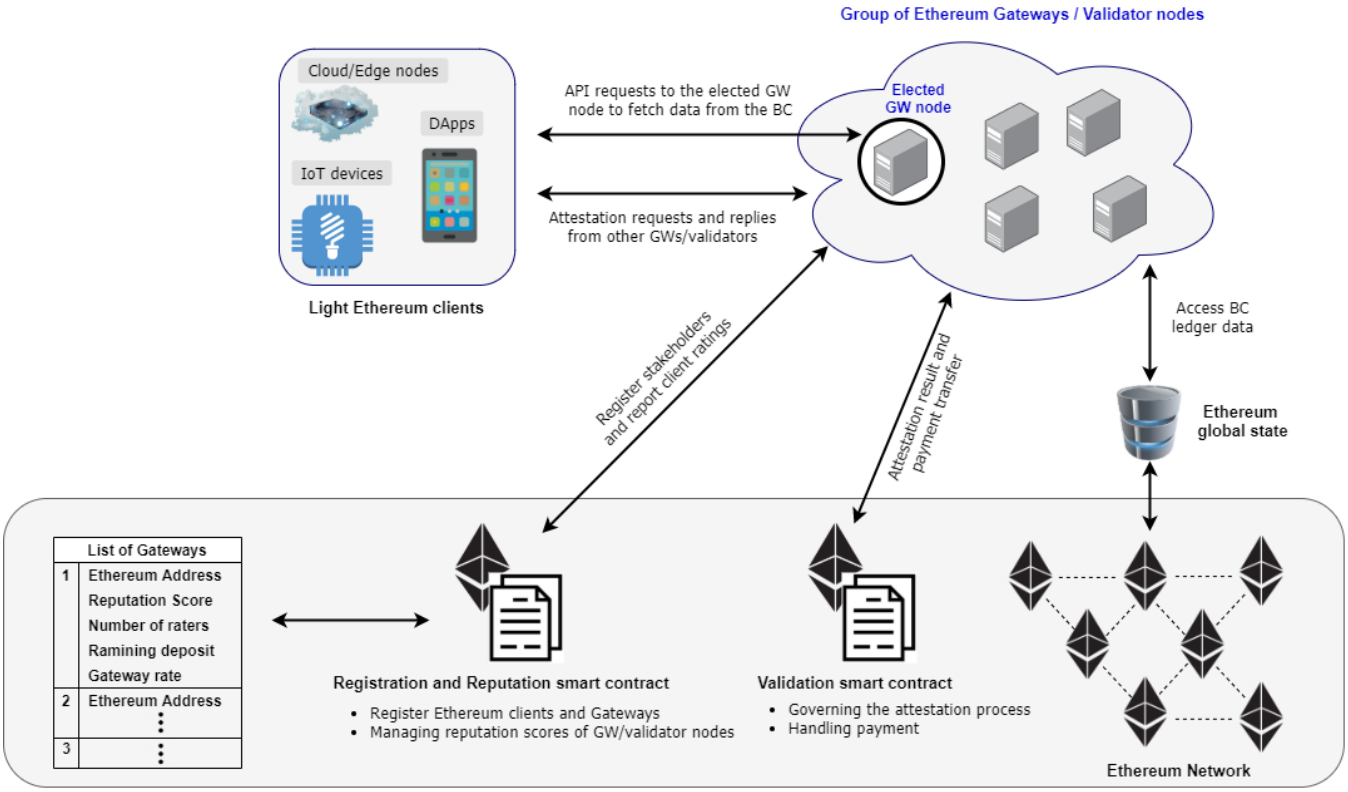

FIGURE 2: Architecture of proposed solution for attesting data by Ethereum blockchain gateways

feedback, missing, and late responses result in losing credibility and balance deductions. Eventually, such nodes can lose all their balance as well as being denied from providing attesting services after continuous loss of performance.

- **Gateways:** In addition to validation, full nodes can serve as a gateway to lightweight nodes. Light clients typically communicate with only one full node to retrieve information about the blockchain but connect to several validators at once. Similar to the validation process, gateways are also rated based on their performance. If gateways fail to provide honest responses, they could also suffer from the loss of their deposit. This could happen if enough validators agree to dispute the response reported by the gateway.
- **Smart contracts:** All aforementioned interactions between light nodes and full nodes are regulated by Ethereum smart contracts. Smart contracts handle the registration of each of the participating entities. This is done to restrict access to specific functions for certain types of users. Furthermore, some functions can only be triggered by users that own that data. For instance, to flag a transaction as attested, the function caller is required to be a registered validator. The Ethereum client is not allowed to trigger this function otherwise. This validator is only allowed to do so for attestation requests that are assigned to it prior to this transaction. All of this data is tracked by the Ethereum address of the caller and the request number that is generated by the smart contract itself. For separation of concerns, the functionality of the system is split into two smart contracts. One of the smart contacts handles registration and keeping track of different entities, their reputation scores, and all data regarding the users. The other smart contract governs the validation process that comprises requests, responses, assigning validators, disputes, and ratings.

V. IMPLEMENTATION DETAILS OF THE PROPOSED SOLUTION

This section presents the implementation details and the algorithms developed for implementing our proposed solution for attesting information returned by Ethereum gateways. We deploy the aforementioned solution on a test Ethereum network virtually for functionality validation and performance evaluation. Remix IDE was used for this purpose as it provides a suitable development environment for development using Solidity language. As previously mentioned, our solution includes two types of smart contracts: the Registration and Reputation smart contract, and Validation smart contract. Following is a brief explanation of the functionality and deployment details of each of those two contracts.

- **Registration and Reputation smart contract:** This contract is deployed once on the Ethereum blockchain

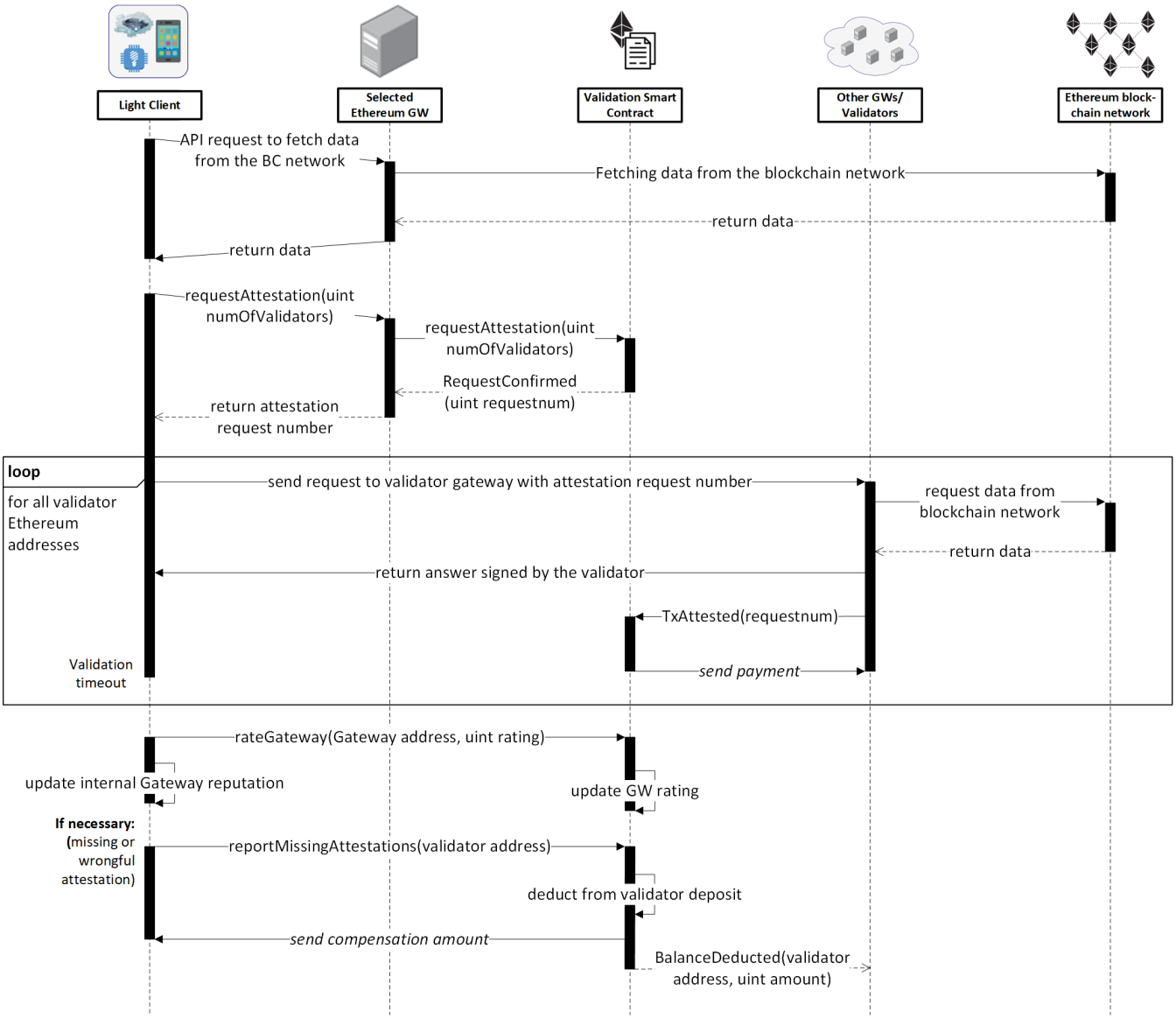

FIGURE 3: Interactions between different stakeholders showing attestation requests and responses.

and given an Ethereum address that other clients use to contact it. Remix IDE provides several virtual accounts or Ethereum addresses for testing purposes, and one of those accounts was utilized to deploy the smart contract. As the name suggests, this smart contract registers all of the entities involved in the system and tracks their data as they are modified throughout the process. The addresses of the owner of the smart contract, the gateways/validators, and the light clients are all recorded via this smart contract. In addition to their addresses, the smart contract records the deposit, rating, and the number of clients who rated every registered full node. These records are valuable and will be used later for deducing the total number of available validators, applying the selection mechanism for validators, deducting some of the balance of misbehaving validators, and other services provided by the system. For instance, when penalizing a gateway for providing false information, the current balance is inspected, and the gateway could be blocked if it does not have enough balance. This smart contract implements payable functions that Ethereum clients can access to register in the smart contract by paying an admission fee that is pre-determined by the contract owner. The penalization and rewards amounts are also stated in this contract. Further, confidential information is protected using private data types and customized *setters* and *getters* are devised to regulate access to such data.

- **Validation smart contract:** Similar to the Registration and Reputation smart contract, this contract is also deployed once using a virtual Ethereum account. The address of the Registration contract is embedded in the

Algorithm 1: Client and Gateway Registration

Input: Client registration fee, Gateway Deposit
Modifier: NA
`/* Client Registration */`
The client transfers the registration fee to the smart contract.
if *Ethereum address of sender has not been previously registered by another entity* $\wedge$ *transferred fee is sufficient* **then**
    Accept payment.
    Append client Ethereum address to the list.
else
    Reject registration.
end
`/* Gateway Registration */`
The client deposits collateral in the smart contract balance.
if *Ethereum address of the gateway is not registered* $\wedge$ *the deposit amount is equal to or more than the minimum deposit value* **then**
    Accept deposit.
    Assign an initial reputation score to the gateway.
    Map the Ethereum address of the gateway to its metadata.
    Increment the total number of gateways available.
else
    Reject registration.
end

Algorithm 2: Attestation request

Input: number of required validators
Modifier: onlyClient
Retrieve number of available validators from Registration contract.
if *Sufficient number of validators is available* $\wedge$ *Correct amount of Ethers has been transferred to the smart contract* **then**
    Generate a new request number.
    Initiate validation request recording the client address, requested validators, and time of request.
    Reserve requested number from the pool of available validators.
    foreach *requested validator* **do**
        Retrieve the validator's Ethereum address.
        Map the validator's address to the request number.
        Send validation assignment to both the light client and the validator.
    end
    Request confirmation.
else
    Reject attestation request.
end

validation contract to refer to the data of registered users. This is done in the constructor when deploying the validation contract. Light clients contact this smart contract to request validation services. The smart contract generates a request number to link it to the request details, including the light client address, validation required, and the validation progress. In addition, these requests are mapped to the assigned validators' addresses. After the request generation, all interactions are restricted to the requester and assigned validators. For instance, only the assigned validator can provide feedback on that request. The validation progress is updated as this feedback arrives from the validators that get paid the agreed fee once they provide their attestation. Furthermore, this smart contract governs disputes and disagreements between clients and validators. Whenever a validator provides false claims or fails to provide feedback, the client reports it, which results in penalizing the former by reducing its credibility score in the previous contract.

Fig. 3 shows how light clients interact with full nodes and smart contracts. As illustrated in this figure, a light client requests specific data from its gateway before requesting attestation from validators. To validate this information, the light client submits an attestation request to the smart contract with the number of requested validators. The client does not contact the validators directly since the selection of nodes to act as validators is made only by the smart contract itself. Therefore, a potentially malfunctioning node cannot influence other validators or select a validator to participate in a conspiracy with it as they do not know the validators assigned to the attestation request prior to the request submission by the client. Subsequently, the smart contract assigns a unique number for the attestation request and confirms the request by triggering an event containing the address of the requester and the confirmation number. In addition, the smart contract assigns the required number of validators to the request and informs all participating entities of this. The light client then proceeds to send the assigned validator its request, to which the latter provides its response. Along with responding to the client, the validator also updates the smart contract of the attestation success. The smart contract provides the payment that has been agreed upon in the smart contract to the validator. Upon receiving attestation confirmations from validators, the smart contract infers the process progress. After the client's timeout, it can report any missing attestations that it did not receive, in addition to dishonest feedback. Finally, clients provide their own feedback about the performance of the gateway and validators via the smart contracts.

The Registration and Reputation smart contract tracks all the entities within the system using their Ethereum Address. Light clients register in the smart contract to access its

Algorithm 3: Confirming attestation request

Input: Request number
Modifier: onlyValidator
if *Request number is valid* $\wedge$ *message sender is assigned to the attached request* **then**
    Transfer payment for validation to the validator.
    Flag the validator as available.
    Update progress of the attestation process.
    if *number of submitted attestations equals the number of requested attestations* **then**
        Announce the completion of the attestation process.
    end
else
    Revert transaction.
end

Algorithm 4: Rating validators

Input: Gateway Address, reputation score
Modifier: onlyClient
Retrieve current reputation score from Regitration and Reputation smart contract using the gateway Ethereum address.
Retrieve number of raters for this gateway.
Set new reputation score = (current score * number of raters + new score) / raters+1
Normalize reputation score to a maximum of 100
if *reputation score* < *minimum_score* **then**
    Block validator.
end
Increment number of raters for this validator.

services by paying a registration fee. As shown in Algorithm 1, the smart contract validates the payment and the Ethereum address and registers the clients. Likewise, gateways also are registered in this smart contract. Gateways deposit an amount as a guarantee for the quality of service. This deposit is deducted as the performance of the node deteriorates. When gateways are first initialized in the smart contract, they are given an initial reputation score that will change upon their interaction with clients. In addition to the reputation score, the number of raters for each specific gateway is recorded and mapped to its Ethereum Address.

Algorithm 2 presents the details for requesting attestation via *requestAttestation()*. This is a payable function, which means that light clients need to attach payment matching the number of validators requested. After performing the required validation, the smart contract generates a new unique request number to track this submitted request. Creating a new random number for each request by hashing algorithms is too expensive. We address this by generating a random number upon deploying the smart contract and incrementing it with each new request. This is done through the validation smart contract, which consults the Registration and Reputation smart contract to get the list of free validators. After reserving the requested number of validators, all involved stakeholders are informed about this assignment individually. Once all stakeholders are notified, the request is confirmed by the smart contract.

Once the validators receive attestation requests, they can are able to retrieve the required information as they have access to the entire blockchain. Algorithm 3 explains how the validation smart contract tracks this interaction and manages the payment to the validator for the attestation service. As Ethereum smart contracts are passive elements, they cannot actively request a response to an attestation request. Instead, it updates the progress of the attestation as the attestation responses arrive from validators. When the responses reach the amount requested by the client, the smart contract announces the completion of the attestation process. In addition, the validators' status is set to *bus* while it is assigned an attestation assignment. After submitting the response, it is flagged as available again to receive other requests.

Light clients can give their feedback about the performance of the gateway/validator it contacted. Algorithm 4 shows how the reputation is updated based on the feedback by clients. It would be ideal for saving all ratings provided by every client that interacted with the gateway or validator. However, we opt for a more cost-efficient approach to save the overall reputation sore and the number of clients that provided that rating. Saving the number of clients has a couple of benefits. First, it is essential to compute the updated reputation score after the client feedback as explained in algorithm 4. Moreover, having recorded feedback by a large group of clients increases the confidence in this validator by other light clients.

Finally, algorithm 5 explains the mechanism of reporting for false information or missing attestations. For instance, if one single validator reported information that does not match other validators, it is assumed that it is trying to cheat or harm honest gateways. Moreover, if the validator fails to reply to the request altogether, it affects the response time of the attestation process and should be penalized for that. This incentivizes gateways and validators to provide honest and timely feedback to avoid reputation deterioration and losing their deposit. However, it would be unfair to promptly block the validator for providing falsified information. This is because the mismatched information could be due to reasons other than trying to deceive the client. For example, the current block number at one of the nodes may not be like the others nodes because it has not synchronized the last block yet. Furthermore, inconsistent information between nodes could be due to each node running a different micro fork and that the information requested comes from a block that has not yet been finalized. These situations are accounted for by penalizing incorrect information. Honest nodes can

Algorithm 5: Report missing or false attestations

Input: Request number, validator address
Modifier: onlyClient
Validate input paramters.
if *Request Number is valid* $\wedge$ *the request belongs to the sender address* $\wedge$ *a minimum time interval has passed since the attestation request* **then**
    Announce the completion of the attestation process.
    if *remaining deposit of validator* <= *penalty amount* **then**
        Deduct the entire validator deposit.
        Remove validator address from the pool of available validators.
        Decrement total number of available validators.
    else
        Deduct penalty amount from the validator's balance.
    end
    Broadcast the amount deducted from the validator.
    Transfer a compensation amount to the light client.
else
    Revert transaction.
end

quickly recover from decreasing their reputation score due to such reasons. Light clients are incentivized to make such reports as they are compensated for such inconsistencies by the validators.

VI. VALIDATION AND EVALUATION

In order to assess the proposed data attestation system, we have used functionality testing, cost analysis and security analysis. This section provides details for all these different aspects of evaluation.

A. FUNCTIONALITY TESTING

This section discusses the validation of the expected outcome for the implemented approach. The code for the solution was implemented, deployed, and tested on Ethereum using Remix IDE. Remix provides a test Ethereum network for deploying and testing smart contracts. Remix also has various plugins that support the debugging of the deployed code, performing unit testing, and conducting a sufficient performance analysis on the code. A log is generated and displayed for each transaction to simulate a real Ethereum network. These logs can be used to explore transactions as they include the inputs, outputs, events triggered, as well as the execution and transaction gas cost of the transaction. In addition, errors and exceptions are also displayed in these logs. Errors include exceeding the gas cost limit, run time errors, and restrictions enforced by the smart contract itself. Such constraints are required to maintain a level of privacy in the system. As such,

FIGURE 4: Gateway/validator newly registered in the Registration and Reputation smart contract.

some methods are restricted to only specific members.

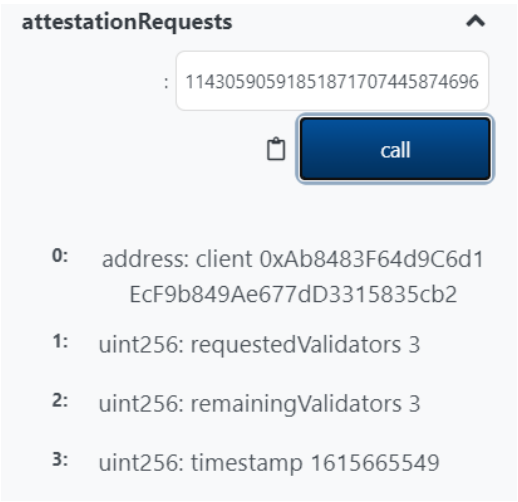

FIGURE 5: The request number mapped to details about the attestation request.

To evaluate the functionality of our smart contracts, we deployed both the Registration and Reputation smart contract at the address 0xc5a98F66719ee680272d8289B8CE227174E2CDDc, and the validation smart contract at the address 0x48ebDb0D8107D12E58266EC9efdc82b047f59FFA. The addresses of both of these contracts are static. In addition, we simulated a complete validation process by creating several validators and clients. These entities are identified by an Ethereum address which maps to their information in the smart contract. For starters, all participating entities are registered by paying the required fee to the smart contract. Fig. 4 shows a newly registered gateway/validator. As as evident from this figure, it has been given an initial reputation score, but no clients have rated it yet.

Once enough entities have registered in the Registration and Reputation smart contract, a client may request attestation

by submitting said request to the validation smart contract. Fig. 5 shows the details of the submitted request. It is given a unique request number that maps, as it can be seen, to the address of the client, time of the request, and other details. These parameters are modified along the process accordingly.

[{ "from": "0x652c9ACcC53e765e1d96e2455E618dAaB79bA595", "topic":
"0x492ad60a65f30e91b3e13c4f5ad4fb603d5e3d21dd6c46498452456aa03e43f5", "event":
"ValidationRequired", "args": { "0":
"114305905918518717074458746961843349530610600381688897462260683236346614152598", "1":
"0x5B38Da6a701c568545dCfcB03FcB875f56beddC4", "2": "0", "requestNumber":
"114305905918518717074458746961843349530610600381688897462260683236346614152598",
"validator": "0x5B38Da6a701c568545dCfcB03FcB875f56beddC4", "validatorNum": "0" } }, {
"from": "0x652c9ACcC53e765e1d96e2455E618dAaB79bA595", "topic":
"0x492ad60a65f30e91b3e13c4f5ad4fb603d5e3d21dd6c46498452456aa03e43f5", "event":
"ValidationRequired", "args": { "0":
"114305905918518717074458746961843349530610600381688897462260683236346614152598", "1":
"0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db", "2": "1", "requestNumber":
"114305905918518717074458746961843349530610600381688897462260683236346614152598",
"validator": "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db", "validatorNum": "1" } }, {
"from": "0x652c9ACcC53e765e1d96e2455E618dAaB79bA595", "topic":
"0x492ad60a65f30e91b3e13c4f5ad4fb603d5e3d21dd6c46498452456aa03e43f5", "event":
"ValidationRequired", "args": { "0":
"114305905918518717074458746961843349530610600381688897462260683236346614152598", "1":
"0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB", "2": "2", "requestNumber":
"114305905918518717074458746961843349530610600381688897462260683236346614152598",
"validator": "0x78731D3Ca6b7E34aC0F824c42a7cC18A495cabaB", "validatorNum": "2" } }, {
"from": "0x652c9ACcC53e765e1d96e2455E618dAaB79bA595", "topic":
"0xb67202e01de5d26f86def24fccaf7364a01b9f5d52d7b122157f5c10907f294f", "event":
"RequestConfirmed", "args": { "0":
"114305905918518717074458746961843349530610600381688897462260683236346614152598", "1":
"0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2", "requestNumber":
"114305905918518717074458746961843349530610600381688897462260683236346614152598",
"clientAddress": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2" } }]

FIGURE 6: A submitted attestation request for three validations.

Following the request submission, the validation smart contract selects the required validators for this request according to the algorithm explained in the previous section. The smart contract then informs the involved stakeholders about this by triggering events. In Fig. 6, 4 events can be seen resulting from the attestation request. The first three *ValidationRequired* events are to inform the client and the validator of assigning the latter to this attestation request. Finally, a *RequestConfirmed* event is triggered in order to confirm the request submission success. These events include parameters such as the request number, client address, and validator information.

[{ "from": "0x1482717Eb2eA8Ecd81d2d8C403CaCF87AcF04927", "topic":
"0x0f95526cfc165dc386d8cb55781bb08e01537989d2a0048eac0cc6a010cf432e", "event":
"AttestationReceived", "args": { "0":
"114305905918518717074458746961843349530610600381688897462260683236346614152598", "1":
"0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db", "requestNumber":
"114305905918518717074458746961843349530610600381688897462260683236346614152598",
"validator": "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db" } }]

FIGURE 7: Validator updating the validation smart contract of attestation completion.

As the validators attest the requests they receive from the light clients, they send back their responses to the clients. Additionally, they update the smart contract by calling the *TxAttested* function and attaching the request number for reference. The smart contract triggers an event to announce the receiving of attestation from the validator, as shown in Fig. 7. Clients can read these events and can follow up with disputes in case they did not match their records.

The smart contracts have built-in exception handling features for catching errors, including exceeding the gas limit, exceeding the smart contract balance, and wrong input parameter formats. In addition, we add extra layers of conditions to maintain restricted access, data privacy, sufficient transfer of payment, and other constraints. Fig. 8 shows an example where a validator triggers the *TxAttested* function call for a request that he was not assigned to. The smart contract reverts the function call and triggers an error regarding that.

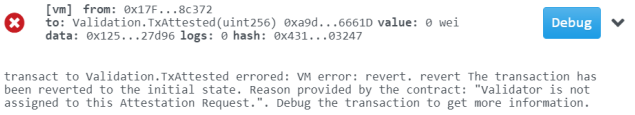

FIGURE 8: Error message showing that the validator is not assigned for the this attestation request.

B. EVALUATION AND ANALYSIS

This section presents a cost analysis of the developed solution and discusses the security aspect of the system. In addition, the generalization of the system is discussed in addition to a comparative analysis with other solutions as well as the challenges faced by such an approach.

1) Cost Analysis

The cost of function calls to Ethereum smart contracts is calculated in gas. The gas that is spent by the function is proportional to its computational complexity. The number of operations in addition to the inputs and outputs of the function sum up to the gas cost of the function. The gas price, however, is a value measured in Ethers set by the clients themselves that refers to how much they are willing to spend for any transaction mined. Naturally, miners choose transactions with higher gas prices to maximize their profit. As a result, the transactions with higher gas prices tend to be mined first. This is a trade-off that Ethereum clients have to address. In addition to the gas price, Ethereum clients set a gas limit that miners are not allowed to exceed when executing a transaction for them. The transaction and execution costs of any function in the smart contract can be deduced directly from the gas cost and gas price. Therefore, smart contract developers always aim to reduce the complexity of the code in order to reduce the cost of utilizing their solution. In this section, we provide cost estimation for each function call in terms of gas as well as US dollars. Converting to a fiat currency helps users appreciate the feasibility of our solution. Ethereum Gas Station helped estimate this by converting the transaction costs provided by Remix IDE to US dollars [14].

The cost of each function call in the smart contracts is mentioned in table 1. As mentioned before, the gas price is set by the Ethereum client, and therefore there is not constant conversion rate between execution costs in gas and their cost in Ethers. Therefore, the gas station analyzes the most recent blocks and proposes a gas price for slow but cheap execution of functions, average execution, and fast execution. These

TABLE 1: Gas cost of Ethereum functions in USD

Method name	Transaction gas cost	Execution gas cost	Slow execution (USD)	Avg. execution (USD)	Fast execution (USD)
`registerClient`	45518	24246	0.00356	0.00435	0.00632
`registerGateway`	92335	71063	0.01042	0.01274	0.01853
`deductBalance`	40350	16070	0.00236	0.00288	0.00419
`requestAttestation`	240536	219072	0.03213	0.03928	0.05714
`TxAttested`	35177	26793	0.00393	0.00480	0.00699
`rateGateway`	64050	41178	0.00604	0.00738	0.01074
`reportMissingAttests.`	36304	41384	0.00607	0.00742	0.01079

gas prices are 36, 44, 55 Gwei, respectively. However, we do not present these prices in Weis or Ethers but rather convert them to USD. The shown prices for the execution cost of each transaction are based on computed values at the time of writing this paper and can vary in the future. The conversion rate to USD as of May 2020 is used, as it is a more reasonable estimation. We prove that the solution is feasible due to the presented function costs. The execution cost of each function is less than $0.04 for cheap and average execution and less than $0.06 for fast execution.

2) Security Analysis

This part of the analysis evaluates the security of the solution proposed. Since our solution heavily relies on the blockchain technology, it inherits many of its security features. This strengthens the resilience of the proposed system due to the decentralized nature of the blockchain. Some of these features are summarized as follows:

- **Availability:** IoT devices and other lightweight devices are constantly operating and in need of continuous access to the gateways' services. Likewise, they require guaranteed responses from the system that supports their trust in said gateways. This access is endorsed by the complete availability of the Ethereum smart contracts. The blockchain ensures that the smart contracts are available at all their mining nodes, which eliminates the single point of failure. The absence of some of these nodes does not affect the total availability of the system as it is compensated by other miners. A transaction is surely going to be mined and included in a block by one or more of these miners. Consequently, the data stored is also replicated at each node and therefore is proven to be always available for extraction.
- **Non-repudiation:** Transactions on the blockchain ledger are singed cryptographically by the Ethereum client. Any member of the blockchain network is required to do so before sending the transaction to the network. As a result, requesting attestations and validations provided by full nodes cannot be denied by any actor. Every transaction is logged in an immutable ledger. This ledger is duplicated at each node to preserve its integrity. Clients and gateways are required to act honestly as the dispute can be easily resolved by referring back to these logs. Since these records are permanent, both parties are held accountable for all their actions.
- **Authorization:** The ubiquity of IoT devices, mobile applications, and other light nodes that require gateways to access the blockchain makes it increasingly difficult to keep their data private and tamper-proof. Providing accessibility to only authorized members becomes difficult with the high number of members in the system, especially in a public network. Ethereum smart contracts support function modifiers that restrict access to each function to a specific group of privileged users. For instance, modifying an attestation request is only possible for the client that requested it or the validators assigned by the smart contract. This maintains the integrity of the system data.
- **The re-entrancy attack:** Previous versions of the blockchain were susceptible to an attack that targets a vulnerability in the blockchain similar to an attack in 2016 on the DAO [15]. Hence, this attack is also known as the DAO attack. The blockchain was vulnerable to *"call to unknown"*. Fortunately, a hard fork in the blockchain was introduced to overcome this vulnerability.
- **The 51% attack:** Blocks are formed by miners in the blockchain network that aggregate transactions, validates them, and forms them into a block to append to the existing chain. These miners get rewarded for doing so as it is a computationally heavy process due to the consensus protocols in the Bitcoin and Ethereum networks. These blockchain networks operate on the assumption that the hashing power required for the consensus protocol can never be owned by a single entity [16], [17]. Nevertheless, if the majority of the network participants or 51% of them collude together, they can virtually single-handedly manipulate the entire network. Thus, this attack is referred to as the majority attack, which is a breach of the consensus protocol [16].

3) Smart Contract Code Vulnerability Analysis

Several security analysis tools were used to discover vulnerabilities in the developed smart contract code for our trustworthy gateways solution. This is done to append the previous analysis of the general blockchain security features. Although Remix IDE flags errors in the syntax of the smart code and identifies run-time errors, these security tools provide a more extensive analysis of the smart code. The smart contract vulnerability analysis is specific to each smart contract and gives us more insight into the possible

exploits to our solution. Oyente tool was first used to detect vulnerabilities in the smart contracts [18]. Oyente checks the code coverage as well as scans the code for several known security weaknesses. For instance, Oyente checks for possible cases of integer overflow and integer underflow. In addition, it checks for potential callstack depth and re-entrancy attacks, any timestamp dependencies, as well as transaction ordering dependence. For each smart contract in the solution, Oyente checks for these vulnerabilities and generates a security report. This contains a summary of all the aforementioned vulnerabilities and their occurrence in the smart contract code. Fig. 9 shows the security report by Oyente generated for the two smart contracts. After several iterations, we were able to modify the code to be resilient to all threats reported by the tool. The report proves that the developed code does not have any of the mentioned security bugs and that the smart contracts are secure and safe for deployment.

```
INFO:root:contract cont.sol:Registration:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                  99.2%
INFO:symExec:     Integer Underflow:                  False
INFO:symExec:     Integer Overflow:                   False
INFO:symExec:     Parity Multisig Bug 2:              False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:               False
INFO:symExec:     Re-Entrancy Vulnerability:          False
INFO:symExec:   ====== Analysis Completed ======
INFO:root:contract cont.sol:Validation:
INFO:symExec:   ============ Results ===========
INFO:symExec:     EVM Code Coverage:                  94.5%
INFO:symExec:     Integer Underflow:                  False
INFO:symExec:     Integer Overflow:                   False
INFO:symExec:     Parity Multisig Bug 2:              False
INFO:symExec:     Callstack Depth Attack Vulnerability:  False
INFO:symExec:     Transaction-Ordering Dependence (TOD): False
INFO:symExec:     Timestamp Dependency:               False
INFO:symExec:     Re-Entrancy Vulnerability:          False
INFO:symExec:   ====== Analysis Completed ======
```

FIGURE 9: Registration & Reputation and Validation smart contracts vulnerabilities recorded by Oyente analysis tool.

The smart contract code was also analyzed the smart for code vulnerabilities by SmartCheck [19]. SmartCheck is an open-source code analyzer tool for analyzing Ethereum smart contracts. It scans the smart contracts against its knowledge base to discover faults and detect errors in the code. SmartCheck can detect about 50 types of errors with different levels of severity. Fig. 10 shows that the current version of the smart contract code developed for this solution is reported to be free of vulnerabilities.

```
+ @smartdec/smartcheck@2.0.1
added 20 packages from 18 contributors and audited 20 packages in 18.5s

3 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

FIGURE 10: Smart contracts vulnerability analysis by SmartCheck.

4) Generalization

The proposed solution is designed for Ethereum smart contracts to complement the interaction between lightweight clients and their gateways. Other blockchain networks that support smart contracts can use the same logic presented in this paper to implement a system for entrusting the gateways for light clients that do not have the resources to participate in the blockchain network. This can be done by modifying the language of the smart contract to one that is supported by that blockchain network. Moreover, a blockchain network that does not support smart contracts can still benefit from most of the functionalities offered by this solution. The integration between different blockchain networks may not be as seamless, but the clients can definitely benefit from such a system. For instance, Bitcoin does not support smart contracts but can still use Ethereum smart contracts to orchestrate the interaction between clients and gateways on the Bitcoin network. Some overhead would be added due to the usage of two blockchain networks by the clients. Nonetheless, the clients can enjoy trustworthy and safe interactions with their gateways due to the proposed approach.

VII. CONCLUSION

In this paper, we have proposed a solution to establish trust by light clients in the blockchain view provided by the connected gateways. The solution is based on Ethereum smart contracts and provides a way for these light clients to attest the blockchain view of their gateways. Light clients submit an attestation request to the smart contract, which assigns the required set of validators to it. Subsequently, the smart contract oversees the progress of the validation process and paying the validators for their services. The developed smart contracts are available for public access on the mentioned Github repository. The development environment of choice was the Remix IDE as it provides valuable features for developers to write smart contracts and test them. In addition, the developed smart contracts were deployed on a test Ethereum network through Remix IDE to validate their functionality and were found to match the requirements set for the solution. Further analysis was made on the smart contract code as we explored the cost of operation for all the features in terms of gas and USD. Each function was found to cost less than $0.04, considering the conversion rate mentioned previously in the cost analysis section. The security features of our blockchain-based solution were discussed to prove its feasibility. As future work, we are looking to complement the blockchain-based approach with a front-end system. Decentralized Applications (DApps) are being utilized to maintain the decentralization of the solution. Subsequently, the proposed system would be ready to be deployed on the mainnet.

ACKNOWLEDGEMENT

This publication is based upon work supported by Khalifa University of Science and Technology under Award No. CIRA-2019-001.

REFERENCES

[1] S. K. Singh, S. Rathore, and J. H. Park, "Blockiot-intelligence: A blockchain-enabled intelligent iot architecture with artificial intelligence," Future Generation Computer Systems, vol. 110, pp. 721–743, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19316474

[2] M. Debe, K. Salah, M. H. U. Rehman, and D. Svetinovic, "Iot public fog nodes reputation system: A decentralized solution using ethereum blockchain," IEEE Access, vol. 7, pp. 178 082–178 093, 2019.

[3] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, "On blockchain and its integration with iot. challenges and opportunities," Future Generation Computer Systems, vol. 88, pp. 173–190, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17329205

[4] M. Debe, K. Salah, M. H. Ur Rehman, and D. Svetinovic, "Monetization of services provided by public fog nodes using blockchain and smart contracts," IEEE Access, vol. 8, pp. 20 118–20 128, 2020.

[5] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," Future Generation Computer Systems, vol. 82, pp. 395–411, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17315765

[6] J. Arshad, M. A. Azad, M. M. Abdellatif, M. H. U. Rehman, and K. Salah, "Colide: a collaborative intrusion detection framework for internet of things," IET Networks, vol. 8, no. 1, pp. 3–14, 2018.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[8] B. Alangot, D. Reijsbergen, S. Venugopalan, and P. Szalachowski, "Decentralized lightweight detection of eclipse attacks on bitcoin clients," in 2020 IEEE International Conference on Blockchain (Blockchain). IEEE, 2020, pp. 337–342.

[9] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in 24th USENIX Security Symposium, 2015, pp. 129–144.

[10] A. Kiayias, A. Miller, and D. Zindros, "Non-interactive proofs of proof-of-work," in International Conference on Financial Cryptography and Data Security. Springer, 2020, pp. 505–522.

[11] B. Bünz, L. Kiffer, L. Luu, and M. Zamani, "Flyclient: Super-light clients for cryptocurrencies," in 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020, pp. 928–946.

[12] K. Todd, "Merkle mountain range," 2012. [Online]. Available: https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md

[13] D. Letz, "Blockquick: Super-light client protocol for blockchain validation on constrained devices," IACR Cryptol. ePrint Arch., 2019.

[14] "Eth gas station," [Accessed: 14 March 2021]. [Online]. Available: https://ethgasstation.info/

[15] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in International conference on principles of security and trust. Springer, 2017, pp. 164–186.

[16] I.-C. Lin and T.-C. Liao, "A survey of blockchain security issues and challenges." IJ Network Security, vol. 19, no. 5, pp. 653–659, 2017.

[17] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in International conference on financial cryptography and data security. Springer, 2014, pp. 436–454.

[18] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in Proceedings of the 2016 ACM SIGSAC conference on computer and communications security, 2016, pp. 254–269.

[19] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "Smartcheck: Static analysis of ethereum smart contracts," in Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, 2018, pp. 9–16.