Trustworthy Blockchain Gateways for Resource-Constrained Clients and IoT Devices

MAZIN DEBE\textsuperscript{1}, KHALED SALAH\textsuperscript{1}, (Senior Member, IEEE), RAJA JAYARAMAN\textsuperscript{2}, IBRAR YAQOOB\textsuperscript{1}, (Senior Member, IEEE), AND JUNAID ARSHAD\textsuperscript{3}

\textsuperscript{1}Department of Electrical Engineering and Computer Science, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates
\textsuperscript{2}Department of Industrial and Systems Engineering, Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates
\textsuperscript{3}School of Computing and Digital Technology, Birmingham City University, Birmingham B4 7XG, U.K.

Corresponding author: Ibrar Yaqoob (ibrar.yaqoob@ku.ac.ae)

This work was supported by Khalifa University of Science and Technology under Award CIRA-2019-001.

\textbf{ABSTRACT} Resource-constrained blockchain clients and the Internet of Things (IoT) devices pose limitations in terms of processing and storing the entire blockchain ledger and mining blocks. Such clients and devices rely on a view of the blockchain provided by full nodes acting as gateways. However, gateway nodes sometimes can provide a distorted view of the blockchain that makes lightweight clients vulnerable to the 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.

In this paper, we propose a data attestation solution that employs full nodes as validators to attest the responses reported by gateways of lightweight nodes. We leverage smart contracts to give lightweight clients confidence in the data reported as they are unable to validate it from the blockchain network themselves. The system governs the attestation process that comprises 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 evaluate the proposed solution with respect to functionality testing, cost of execution, and security analysis of the developed smart contracts. We make our smart contracts code and testing scripts publicly available.

\textbf{INDEX TERMS} Blockchain, Ethereum, IoT, gateways, trust, lightweight nodes, full nodes.

\section{I. INTRODUCTION}

The wide adoption of blockchain technology has resulted in the growing use of lightweight clients to support integration between blockchain and the Internet of Things (IoT). The integration of these technologies has facilitated the 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].

The primary motivation in using IoT devices is their ability to provide lightweight and 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 utilizing 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 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 lightweight nodes to the

actual blockchain. Generally, lightweight 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 lightweight 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 blockchain network. Although lightweight clients do not require to run consensus algorithms, append blocks and transactions to the existing chain. However, they only require to submit transactions, and in some cases, make decisions and transactions, respectively. In this paper, a data attestation approach is presented, which utilizes 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 lightweight 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 as follows:

- 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 lightweight nodes and assigns respective validators to verify the information reported by the gateway.
- We present detailed descriptions 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. We make our smart contracts code and testing scripts publicly available.
- 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.

The remainder of this paper is organized as follows: Section II discusses background information about 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 along with cost and security analyses. We provide our concluding remarks in Section VII.

### II. BACKGROUND INFORMATION

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

#### A. BLOCKCHAIN TECHNOLOGY

A blockchain network 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 append blocks and transactions to the existing chain. Regular blockchain clients do not require to run consensus algorithms, but they only require to submit transactions, and in some blockchain networks, pay fees for the transactions. 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 [9]. 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, lightweight 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 [10]. 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 discuss the existing solutions that aim to address the threat of eclipse attacks on lightweight clients connecting to gateways that provide 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 lightweight blockchain clients. However, the gossip protocol is computationally expensive for lightweight clients with minimal processing power.

Kiayias et al. proposed an improvement to the current Proof of Work (PoW) consensus protocol [11]. 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 lightweight blockchain clients in terms of resources and processing ability in contrast to the traditional Simplified Payment Verification (SPV). In SPV, lightweight 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 [12] proposed 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) [13]. 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 [14] comprising a super-light 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

Herein, we propose a data attestation solution that leverages Ethereum smart contracts to validate the information provided by blockchain gateways to lightweight nodes, as shown in Fig. 2.
Lightweight clients often connect to one gateway. Although this allows resource-constrained devices to utilize blockchain technology, it compromises one of the prime benefits of using the blockchain, i.e., decentralization. Reliance on a single element to provide access to the entire network contradicts the consensus protocols. Our solution relies on using the Ethereum-based smart contracts that manage attestation of the data provided by the gateway. Privacy of information is maintained nevertheless through restricted access to functions and data by users. All stakeholders are registered through the smart contract and are given an Ethereum address acting as an identifier to track the process of attestation and solve disputes.

Lightweight Ethereum clients contact Ethereum gateways that act as full nodes to reference their copy of the blockchain. These clients request data from the gateway nodes through API requests. Full nodes can 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 lightweight nodes sometimes need to attest the 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 lightweight 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. Lightweight 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 to 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 an 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.

Lightweight 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 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 are 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’s 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 with each other. The role of each one is explained further below.

- **Lightweight Clients:** These lightweight nodes include decentralized mobile applications, IoT devices, or any other resource-constrained device. Limited storage and processing power inhibit them from downloading the entire blockchain and making transactions independently. 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, lightweight clients can also request attestation from other full nodes to validate the information.

- **Validators:** Full nodes provide attestation services to lightweight clients and get rewarded in *ethers* for that service. The performance of these validators is monitored by clients themselves that provide their evaluation. False 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 the continuous loss of performance.

- **Gateways:** In addition to validation, full nodes can serve as a gateway to lightweight nodes. Lightweight 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 lightweight 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 implementation details of algorithms developed for attesting information returned by Ethereum gateways. We deploy our solution on a test Ethereum network virtually for functionality validation and performance evaluation. We use Remix IDE to provide a suitable development environment using Solidity language. The proposed solution includes two types of smart contracts; namely, the Registration and Reputation smart contract and Validation smart contract. In the following subsections, we explain them in more detail.

- **Registration and Reputation Smart Contract:** This contract is deployed once on the Ethereum blockchain 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 lightweight 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
FIGURE 3. Interactions between different stakeholders showing attestation requests and responses.

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 predetermined 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 validation contract to refer to the data of registered users. This is done in the constructor when deploying the validation contract. Lightweight 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 lightweight 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.

Algorithm 1: Client and Gateway Registration

```plaintext
Input: Client registration fee, Gateway Deposit

// Client Registration
/*
1 The client transfers the registration fee to the smart contract.
2 if Ethereum address of sender has not been previously registered by another entity and transferred fee is sufficient then
3 Accept payment.
4 Append client Ethereum address to the list.
5 else
6 Reject registration.
7 end

// Gateway Registration */
8 The client deposits collateral in the smart contract balance.
9 if Ethereum address of the gateway is not registered and the deposit amount is equal to or more than the minimum deposit value then
10 Accept deposit.
11 Assign an initial reputation score to the gateway.
12 Map the Ethereum address of the gateway to its metadata.
13 Increment the total number of gateways available.
14 else
15 Reject registration.
16 end
```

Algorithm 2: Attestation Request

```plaintext
Input: number of required validators

// onlyClient
1 Retrieve number of available validators from Registration contract.
2 if Sufficient number of validators is available and Correct amount of Ethers has been transferred to the smart contract then
3 Generate a new request number.
4 Initiate validation request recording the client address, requested validators, and time of request.
5 Reserve requested number from the pool of available validators.
6 foreach requested validator do
7 Retrieve the validator’s Ethereum address.
8 Map the validator’s address to the request number.
9 Send validation assignment to both the lightweight client and the validator.
10 end
11 Request confirmation.
12 else
13 Reject attestation request.
14 end
```

Fig. 3 shows how lightweight clients interact with full nodes and smart contracts. As illustrated in this figure, a lightweight client requests specific data from its gateway before requesting attestation from validators. To validate this information, the lightweight 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 lightweight 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 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. Lightweight clients register in the smart contract to access its 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 lightweight 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.

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

VI. VALIDATION AND EVALUATION

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 subsection presents the details for testing the different functionalities of the discussed solution. The developed smart contracts were tested in the same environment that was used for development and deployment, Remix IDE. It offers a test Ethereum network to explore and simulate different
Algorithm 5: Report Missing or False Attestations

\textbf{Input:} Request number, validator address

1. Modifier: onlyClient

2. Validate input parameters.

3. If Request Number is valid and the request belongs to the sender address and a minimum time interval has passed since the attestation request then

   4. Announce the completion of the attestation process.

   5. If remaining deposit of validator \(\leq\) penalty amount then

      6. Deduct the entire validator deposit.

      7. Remove validator address from the pool of available validators.

      8. Decrement total number of available validators.

   else

      9. Deduct penalty amount from the validator’s balance.

10. End

11. Broadcast the amount deducted from the validator.

12. Transfer a compensation amount to the lightweight client.

13. Else

14. Revert transaction.

15. End

scenarios. Also, debugging and performance evaluation can be conducted on this platform. This is done for any transaction that is executed as they are recorded in logs that contain details about the function parameters, cost, triggered events, and errors. This simulates to a great extent the Ethereum network and its behavior after deploying the smart contracts on the mainnet. In addition to Ethereum network constraints on the function calls, the customized restrictions by the smart contract are also embedded in those logs. These include privileged access to certain functions, wrong payments, and wrong inputs. Both the Registration and Validation smart contracts were deployed in the beginning to start the testing process. Moreover, several validators and clients were registered in the relevant smart contract using their Ethereum address. These actors are registered by paying the required fee to the smart contract. 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 a 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.

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.

As the validators attest the requests they receive from the lightweight 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.

Clients can rate their experience with the validators as mentioned before by providing a rating score. This results in updating the validator’s score recorded by the smart contract. Fig. 9 shows the log output after a client has provided its feedback about a validator. This log shows the validator Ethereum address, the previous rating of the validator prior to the feedback, in addition to the new updated score post the feedback. Since all the required data is recorded on the blockchain, clients can verify that their feedback has been recorded and directly affected the rating of the validator.

B. EVALUATION AND ANALYSIS

This section presents cost analysis and discusses the security aspect of the system. Also, the generalization of the proposed system is discussed.

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 [15].

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 thus 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, average execution, and fast execution. These 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 slow and average execution and less than $0.06 for fast execution.

2) SECURITY ANALYSIS

Herein, we evaluate the security features of the proposed solution. Since our solution heavily relies on blockchain technology, it inherits many of its security features. This strengthens the resilience of the proposed system because of the decentralized nature of the blockchain. Some of the key features are summarized as follows:

FIGURE 6. A submitted attestation request for three validations.

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

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

FIGURE 9. Rating of a validator being updated after feedback from a client.
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            | 40550                | 16570              | 0.003036             | 0.00238              | 0.00419              |
| requestAttestation       | 240536               | 219072             | 0.03213              | 0.03928              | 0.05714              |
| rxAAttested              | 35177                | 26793              | 0.00393              | 0.00480              | 0.00699              |
| rateGateway              | 64050                | 41178              | 0.00604              | 0.00738              | 0.01074              |
| reportMissingAttestests. | 36304                | 41384              | 0.00607              | 0.00742              | 0.01079              |

- **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 signed 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 lightweight 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 [16]. 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.

3) **SMART CONTRACT CODE VULNERABILITY ANALYSIS**

Multiple security analysis tools were used to discover vulnerabilities in the developed smart contract code for our trustworthy gateway 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 [19]. 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. 10 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.
The smart contract code was also analyzed for code vulnerabilities using SmartCheck [20], which is an open-source code analyzer tool. 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. 11 shows that the current version of the smart contract code developed for this solution is reported to be free of vulnerabilities.

**FIGURE 10.** Registration & reputation and validation smart contracts vulnerabilities recorded by Oyente analysis tool.

**FIGURE 11.** 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 lightweight 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 get benefits 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 for lightweight clients to attest the blockchain view of their gateways. The lightweight clients submit the attestation request to one of the smart contracts, which assigns the required set of validators. The developed smart contract oversees the progress of the validation process and assists in paying the validators against their services. The developed smart contracts code is made publicly available on the GitHub repository. We used Remix IDE as it provides valuable features in terms of writing and testing smart contracts. Also, the developed smart contracts were deployed on a test Ethereum network 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 functions. Each function was found to cost less than $0.04, considering the conversion rate mentioned in the cost analysis section. The security features of the proposed blockchain-based solution were presented to show that it is secure enough against well-known attacks and vulnerabilities. In the future, we aim to complement our blockchain-based approach with a front-end system. Decentralized Applications (DApps) will be utilized to maintain the decentralization of the solution. Subsequently, the proposed system would be ready to be deployed on the mainnet.

**REFERENCES**

[1] S. K. Singh, S. Rathore, and J. H. Park, “BlockIoTelligence: A blockchain-enabled intelligent IoT architecture with artificial intelligence,” *Future Gener. Comput. Syst.*, vol. 110, pp. 721–743, Sep. 2020.

[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. 178082–178093, 2019.

[3] A. Reyna, C. Martín, J. Chen, E. Soler, and M. Díaz, “On blockchain and its integration with IoT. Challenges and opportunities,” *Future Gener. Comput. Syst.*, vol. 88, pp. 173–190, Nov. 2018.

[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. 20118–20128, 2020.

[5] M. A. Khan and K. Salah, “IoT security: Review, blockchain solutions, and open challenges,” *Future Gener. Comput. Syst.*, vol. 82, pp. 395–411, May 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17315765

[6] J. Arshad, “COLIDE: A collaborative intrusion detection framework for Internet of Things,” *IET Netw.*, vol. 8, pp. 3–14, Jan. 2019.

[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 *Proc. IEEE Int. Conf. Blockchain*, Nov. 2020, pp. 337–342.

[9] I. Yaqoob, K. Salah, R. Jayaraman, and Y. Al-Hammadi, “Blockchain for healthcare data management: Opportunities, challenges, and future recommendations,” *Neural Comput. Appl.*, vol. 4, pp. 1–16, Jan. 2021, doi: 10.1007/s00521-020-05519-w.

[10] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, “Eclipse attacks on bitcoin’s peer-to-peer network,” in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 129–144.

[11] A. Kiyas, A. Miller, and D. Zindros, “Non-interactive proofs of proof-of-work,” in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Cham, Switzerland: Springer, 2020, pp. 505–522.

[12] B. Bunz, L. Kiffer, L. Lau, and M. Zaman, “FlyClient: Super-light clients for cryptocurrencies,” in *Proc. IEEE Symp. Secure. Privacy (SP)*, May 2020, pp. 928–946.
[13] K. Todd. (2012). Merkle Mountain Range. [Online]. Available: https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md

[14] D. Letz, “BlockQuick: Super-light client protocol for blockchain validation on constrained devices,” in Proc. IACR, 2019, p. 579.

[15] Etr Gas Station. Accessed: Mar. 14, 2021. [Online]. Available: https://ethgasstation.info/

[16] N. Atzei, M. Bartoletti, and T. Cimoli, “A survey of attacks on Ethereum smart contracts (SoK),” in Proc. Int. Conf. Princ. Secur. Trust. Berlin, Germany: Springer, 2017, pp. 164–186.

[17] I. C. Lin and T. C. Liao, “A survey of blockchain security issues and challenges,” Int. J. Netw. Secur., vol. 19, no. 5, pp. 653–659, 2017.

[18] I. Eyal and E. G. Sirer, “Majority is not enough: Bitcoin mining is vulnerable,” in Proc. Int. Conf. Financial Cryptography Data Secur. Berlin, Germany: Springer, 2014, pp. 436–454.

[19] S. Badruddoja, R. Dantu, Y. He, K. Upadhayay, and M. Thompson, “Making smart contracts smarter,” in Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC), May 2021, pp. 254–269.

[20] S. Tikhomirov, E. Vokresenskaya, I. Vinitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, “SmartCheck: Static analysis of Ethereum smart contracts,” in Proc. 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain, May 2018, pp. 9–16.

MAZIN DEBE received the B.Sc. degree in computer engineering and the M.Sc. degree in electrical and computer engineering from Khalifa University of Science and Technology, Abu Dhabi, United Arab Emirates. He is currently working with the Center for Cyber-Physical Systems, Khalifa University of Science and Technology, as a Research Associate. He has published multiple research articles in highly ranked IEEE conferences and journals. His research interests include blockchain technology, the Internet of Things (IoT), fog computing, and supply chain applications.

RAJA JAYARAMAN received the bachelor’s and master’s degrees in mathematics in India, the Master of Science degree in industrial engineering from New Mexico State University, and the Ph.D. degree in industrial engineering from Texas Tech University. He is currently an Associate Professor with the Department of Industrial and Systems Engineering, Khalifa University, Abu Dhabi, United Arab Emirates. His expertise is in multi-criteria optimization techniques applied to diverse applications, including supply chain and logistics, healthcare, energy, environment, and sustainability. His postdoctoral research was centered on technology adoption and implementation of innovative practices in the healthcare supply chains and service delivery. He has led several successful research projects and pilot implementations in the area of supply chain data standards adoption in the U.S. healthcare system. His research interests include blockchain technology, systems engineering and process optimization techniques to characterize, model and analyze complex systems with applications to supply chains, maintenance operations planning, and healthcare delivery. His research has appeared in top-rated journals, including Annals of Operations Research, IIE Transactions, Energy Policy, Applied Energy, Knowledge-Based Systems, IEEE ACCESS, Journal of Theoretical Biology, Engineering Management Journal, and others.

KHALED SALAH (Senior Member, IEEE) received the B.Sc. degree in computer engineering with a minor in computer science from Iowa State University, USA, in 1990, and the M.S. and Ph.D. degrees in computer systems engineering from Illinois Institute of Technology, USA, in 1994 and 2000, respectively. He is currently a Full Professor with the Department of Electrical and Computer Engineering, Khalifa University, United Arab Emirates. He is currently leading a number of projects on how to leverage blockchain for healthcare, 5G networks, combating deepfake videos, supply chain management, and AI. He has over 220 publications and holds three U.S. patents, has been giving a number of international keynote speeches, invited talks, tutorials, and research seminars on the subjects of blockchain, the IoT, fog and cloud computing, and cybersecurity. He served as the Chair of the Track Chair for IEEE GLOBECOM 2018 on Cloud Computing. He is an Associate Editor of IEEE Blockchain Tech Briefs and a member of IEEE Blockchain Education Committee.

IBRAR YAQOOB (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Malaya, Malaysia, in 2017. He is currently working with the Department of Electrical Engineering and Computer Science, Khalifa University, United Arab Emirates. Previously, he worked as a Research Professor with the Department of Computer Science and Engineering, Kyung Hee University, South Korea, where he completed his postdoctoral fellowship under the prestigious grant of Brain Korea 21st Century Plus. He worked as a Researcher and a Developer with the Centre for Mobile Cloud Computing Research (C-MC2R), University of Malaya. His numerous research articles are very famous and among the most downloaded in top journals. He has been listed among top researchers by Thomson Reuters (Web of Science) based on the number of citations earned in six categories of computer science. He has been involved in a number of conferences and workshops in various capacities. His research interests include big data, blockchain, edge computing, mobile cloud computing, the Internet of Things, healthcare, and computer networks. He is currently serving/served as a guest/associate editor for various journals.

JUNAID ARSHAD received the Ph.D. degree in computer security from the University of Leeds, U.K., in 2011. He is currently an Associate Professor with the School of Computing and Digital Technology, Birmingham City University, U.K. He has been actively involved in publishing high quality research within cybersecurity. He has successfully published at high quality venues, including journals, book chapters, conferences, and workshops. His research interests include investigating security challenges for diverse computing paradigms, such as distributed computing, cloud computing, the IoT, and distributed ledger technologies. He is an Associate Editor of the Cluster Computing and IEEE ACCESS journals. He regularly serves on program and review committees for several journals and conferences.

* * *