sustainability

MDPI

Article

Robust Decentralized Proof of Location for Blockchain Energy Applications Using Game Theory and Random Selection

Yaçine Merrad [1], Mohamed Hadi Habaebi [1,*], Md. Rafiqul Islam [1], Teddy Surya Gunawan [2] and Mokhtaria Mesri [3]

[1] IoT & Wireless Communication Protocols Laboratory, Department of Electrical & Computer Engineering, International Islamic University Malaysia, Kuala Lumpur 53100, Malaysia; yacinechoupot@yahoo.fr (Y.M.); rafiq@iium.edu.my (M.R.I.)
[2] Department of Electrical & Computer Engineering, International Islamic University Malaysia, Kuala Lumpur 53100, Malaysia; tsgunawan@iium.edu.my
[3] Department of Electronics, University Amar Télidji of Laghouat, Laghouat 03000, Algeria; m.mesri@lagh-univ.dz
* Correspondence: habaebi@iium.edu.my

Abstract: To combat the problem of illegal access to a service, several location proof strategies have been proposed in the literature. In blockchain-based decentralized applications, transactions can be issued by IoT nodes or other automated smart devices. Key pair encryption and private key signing have been defined mainly for human identification in blockchain applications, where users are personally and responsibly concerned about the confidentiality of their private key. These methods are not suitable for computing nodes whose private key is implemented in the software they run. Ensuring that transactions are issued by a legitimate sender with the proper credentials is a bigger concern in applications with financial stakes. This is the case with blockchain energy trading platforms, where prosumers are credited with tokens in exchange for their contributions of energy. The tokens are issued by smart meter nodes installed at fixed locations to monitor the energy inputs and outputs of a given prosumer and claim energy tokens on its behalf from a defined smart contract in exchange for the energy it feeds into the grid. To this end, we have developed a decentralized Proof-of-Location (PoL) system tailored to blockchain applications for energy trading. It ensures that automated transactions are issued by the right nodes by using smart contract-based random selection and a game-theoretic scenario suitable for blockchain energy trading.

Keywords: blockchain; energy trading; game theory; proof of location; random selection

Citation: Merrad, Y.; Habaebi, M.H.; Islam, M.R.; Gunawan, T.S.; Mesri, M. Robust Decentralized Proof of Location for Blockchain Energy Applications Using Game Theory and Random Selection. *Sustainability* **2022**, *14*, 6123. https://doi.org/10.3390/su14106123

Academic Editor: Luis Hernández-Callejo

Received: 16 April 2022
Accepted: 16 May 2022
Published: 18 May 2022

1. Introduction

When the blockchain first appeared, it served as the foundation for cryptocurrency applications. Financial transactions were recorded in the public ledger. The validity of data uploaded to the blockchain depended on the validity of the financial transaction (no double spending). This was ensured by models such as the unspent transaction output (UTXO) model in Bitcoin and the balance-based account model in Ethereum, which work by making each account's transaction history accessible and thus the validity of a new transaction can be verified.

Key pair cryptography and digital signatures are used to verify the identity of the issuer of the transaction. This worked well when blockchain was only meant for financial cryptocurrency transactions, and transactions were issued by mindful human beings who personally cared about maintaining the confidentiality of their secret key. However, with the advent of smart contracts, blockchain has evolved beyond its original application in finance and now enables decentralized applications in a variety of areas, particularly the Internet of Things (IoT), where systems are automated and enabled through the decentralized execution of smart contracts without the need for a server or centralized cloud entity.

In such scenarios, IoT nodes can invoke smart contract functions that are blockchain transactions, based on defined code implemented on the IoT nodes. Because the identity of the transaction issuer is defined in the software running on the IoT node, and is thus easily accessible, private key encryption cannot guarantee the identity of the transaction issuer in such cases. Even if it is closed-source code, it is not secret to the developer of the code, who, in turn, can share it with others. Although misuse of an IoT or smart node identity is not always mandatory for some IoT applications, it is so for others. A typical example is the peer-to-peer (P2P) energy trading system, where prosumers are rewarded with tokens for the energy they generate and feed into the grid. An assigned smart meter measures and monitors the energy fed into the grid and is responsible for claiming a token on behalf of the prosumer. Such architectures have already been proposed in [1–3]. In such a situation, it is tempting for prosumers to abuse the identity of their assigned smart meter to claim an undeserved energy token. Therefore, it is critical to include a verifiable identity attribute that is difficult for others to guess. As smart meter nodes are permanently deployed, a location trace may be sufficient to identify the origin of the transaction. In this paper, we describe a decentralized PoL, suitable for blockchain-based energy trading. Section 2 reviews previous work on the blockchain PoL and highlights the challenges of implementing a reliable decentralized PoL. Section 3 describes the implementation of the blockchain energy trading system to which the PoL is relevant. Section 4 describes the proposed PoL approach and explains how it eliminates all fraud and collusion possibilities. Section 5 presents the obtained results, and Section 6 presents conclusions.

2. Background and Related Work

2.1. Blockchain

The blockchain was originally created to enable the first decentralized cryptocurrency [4], where there is no need for central authority such as a bank to validate or deny transactions. Since then, it has seen tremendous growth to implement any decentralized application, revolutionizing various processes and sectors. Blockchain is implemented on a peer-to-peer network where there is no point of failure [5,6]. Further, it combines hashing technology, mainly the SHA256 hashing algorithm, digital signature, and consensus algorithm [7]. After the 2008 financial crisis, cryptocurrency, on which the blockchain originated, required every transaction that is issued to be propagated to all peers on the network in the hope that it will be confirmed by consensus [4,8]. Transactions take the form of a data structure containing the addresses of the issuer and the recipient, the number of coins to be transferred, and the issuer's digital signature to tag messages in a way that uniquely identifies the signer [9]. When any entity connecting to the blockchain (node) joins the blockchain network and creates an account, the system generates a unique key pair using a well-known mathematical model, namely elliptic curve cryptography [10]: a public key, which is publicly known and essential for pseudo-anonymous identification of the user's account, and a private key, which is used to sign the transaction and is kept confidential for authentication and encryption [11]. The security of this system stems from the fact that a particular transaction can only be issued on behalf of an account if the issuer possesses the private key of that account [4]. In addition, the blockchain is a tamper-proof structure, so any data or transactions committed to the blockchain can never be deleted, denied, or modified. All nodes in the network should have an identical copy of the most recent blockchain ledger. Based on this, the blockchain contains the history of all issued transactions so that the balance of all accounts can be traced and is known to all nodes in the network. Blockchain, as a tamper-proof structure, is based on two principles [7]:

- The first principle: Blocks are chained together by their hashes. The hash is the difficult mathematical problem that miners must solve to find a block. Usually, the SHA256 hash algorithm is used [12]. This goes according to main principles, which are: (i) For two different inputs, never obtain the same output. (ii) For a given output, it is theoretically impossible to obtain the corresponding input. Thus, each block is

hashed with the hash of the previous block, so that chain tampering is detectable unless the hashes are changed appropriately.

- The second principle: The tamper-proof structure of the blockchain is based on the principle "the longest chain wins". Consequently, there are conditions attached to the block's hash and readability that make adding a new block difficult and resource intensive.

Moreover, adding new valid blocks is rewarded. The first account to add a new valid block is accepted and receives a coin reward, making the process of adding new blocks a race. Whereas a fraudulent node is tempted to disrupt the current chain, other nodes have already added new valid blocks and created a new, longest chain. Blockchain technology reached a key milestone when Satoshi Nakamoto introduced the idea of Bitcoin as a cryptocurrency [4], and many business organizations allowed their customers to use Bitcoin as a payment method. Since then, and with the massive rise in Bitcoin's popularity, a number of other cryptocurrencies have emerged, including Tether, Tezos, Litecoin, Monero, and Maker [13]. After attracting incredible attention by proving its efficiency behind several cryptocurrencies that have market capitalization in the billions, and after more than a decade of adoption, the blockchain concept has evolved beyond what it was originally intended for. Blockchain, as the technology behind cryptocurrencies, has evolved into a more global concept to be the technology behind decentralization [5].

We believe it is unfair to talk about blockchain without addressing smart contracts. Smart contracts are computer codes that are executed in a decentralized manner on each node of the network. The state of the smart contract is stored in the public ledger, which consists of the smart contract variables that are updated each time a smart contract function, which is triggered by a smart call, is executed. This allows terms and conditions for agreements between two or more parties to be defined in a decentralized manner [14]. These digitally written lines of code are then uploaded to the blockchain, making them immutable, indisputable, and visible to all parties on the network [15]. Smart contracts are inspired by the Bitcoin scripting concept. Basically, Bitcoin allows adding a script to any transaction for any related specification or condition [16]. A Bitcoin transaction must follow a specific format and contain both an input and an output [4]. For example, suppose Alice sends three bitcoins to Ali; this transaction has three bitcoins as input and mentions as output that these three coins are for Ali. However, if Ali wants to send these bitcoin to another party, he should issue a transaction with an input that references the output of Alice's transaction to show where the coins originated from. Similarly, Alice's input in the transaction intended for Ali points to another transaction output from which the coins were sent to her. This continues until it comes to a coin-base transaction, i.e., the transaction that provides the mining reward without input. This input-output format helps verify any condition or specification set up as a script for the coins before they are used. If Alice added a condition to the transaction stating that these coins can only be spent two months after the transaction date, Ali must reference Alice's transaction output in his transaction input. Anyone can then review the predefined conditions for approving that transaction. The coins cannot be issued until two months have passed since the transaction date.

This concept has since been expanded from simple script forms to advanced algorithmic codes and computer protocols for establishing more complex criteria in the form of contract codes that should enable proper, largely automated workflows. In essence, various industries are investing in blockchain technology, despite the high expectations of academics, which highlights the ongoing hurdles to blockchain adoption, particularly in the energy sector.

2.2. Proof of Location

In recent years, the 'smart city' revolution has paved the way for innovative solutions to improve living standards, centered on the Internet of Things (IoT), leading to real-time interactions [17]. Indeed, the rise of wireless devices has been facilitated by the emergence of a new class of applications known as location based services (LBS). Location based

services is a growing technology that focuses on providing GIS and spatial data through mobile and field devices. Location-based apps and services have gained popularity in recent years due to the variety of useful services they provide. These include activity tracking apps, location-based services, cognitive radio networks (CRNs), and location-based access control systems, which are used in both established and emerging domains. Navigation software, social networking services, location-based advertising, and tracking systems are all examples of location-based services [18,19]. These location-based applications/services may also be used in other access control systems that require a proximity sensor, such as in a critical healthcare system or for entertainment purposes, where these systems may only provide content to users who are in a particular area of interest while charging users who are in a different location [20].

For many current applications, it is critical that users prove their location to the services. Users may misrepresent their location in order to receive rewards [21,22]. To confirm user legitimacy, transparency and control over the use and sharing of location data are required. Depending on the system architecture, the mechanisms of location proof (LP) are divided into two groups: centralized and decentralized. The most recent centralized mechanism was proposed by Javali et al. [23]. In their paper, the authors propose a novel technique to create a location proof for mobile users and to verify the location information by application services. Their approach takes advantage of the unique properties of the WiFi signal and uses a fuzzy vault technique that is potentially secure. However, their work does not consider scenarios in which Access Points (APs) collaborate with adversaries to create a fake location proof or services that deny access to honest users while granting illegal access to dishonest users. In addition, the proposed architecture uses a number of entities, which makes it costly to implement. In decentralized architectures, there is no need to use trusted access points to generate LP. Therefore, these architectures tend to be less costly to develop than centralized mechanisms. In decentralized systems, however, architectural design ingenuity is more challenging to ensure the reliability of the required LPs. Several researchers have studied and proposed various techniques for securing LPs in smart systems, all of which use blockchain smart contracts that have become essential for today's decentralized applications. The initial decentralized architectures presented in the literature face a number of security and privacy issues, such as target-target collusion (terrorist scams) [24–26], target-prover collusion [27], and site privacy threats [28]. In [29], the authors address the threat of target-target collusion by proposing a fixed-time frame for the transmission of LP. Although the ingenuity of this approach, which has definitely reduced the possibility of target-target collusion, is shown by the results obtained by the authors and presented in the paper, such an approach is not infallible, as the execution time of smart contract transactions depends entirely on the time it takes to mine them in a block, which is directly related to the difficulty of block mining in Proof of Work (PoW) consensus-based blockchains.

It is worth noting that reducing the mining difficulty can lead to gaps in the reliability and security of the blockchain. In [30], it is proposed that location claims are curated by tokens. The location claim is subject to a vote that decides the validity of the PL claim, and if the claim has more rejecting nodes than approving nodes, the claimant loses its deployed tokens. This approach also relies on the integrity of the voters, which can be damaged especially if the destination has financial significance. To ensure the reliability of blockchain-based PoL, we believe it must be designed specifically for the application in which it will be used. In this work, we presented a secure yet decentralized PoL designed specifically for blockchain energy trading platforms. In doing so, we used game theory applicable to blockchain energy trading and decentralized random selection based on smart contracts.

The biggest challenge to decentralized PoL is malicious collusion, which can be divided into the following categories:

- Target-target collusion: This occurs when two malicious target nodes collude with each other at different sites. A target node may send a signed PoL request to the

colluding node, which then forwards it to the assigned provers on its behalf. The LP calculated by the provers would be based on the location of the collaborating node and would be appropriate for the second target node.

- Prover-prover collusion: In this situation, a dishonest prover conspires with a remote malicious witness and provides him with a fake location proof. This form of malicious collaboration is most difficult to handle in degenerate untrusted positioning. Most implemented proof of location are centralized, i.e., there are defined anchor nodes that are trusted as location verifiers. There is a growing interest in implementing blockchain-based decentralized location proofs in a trustless architecture. However, as noted in [29], the main challenge in blockchain-based decentralized proof of location is the prover-prover collusion. What we propose is a fully trustless blockchain-based proof of location that is specifically designed for blockchain-based energy applications and robust against prover-prover collusion.

3. Blockchain Enabled P2P Trading Scheme

An ERC20 token smart contract will be used for the proposed blockchain-backed P2P trading system. Tokens are delivered to prosumers when a certain amount of energy is injected into the distribution system. Prosumers receive their earned tokens upon request from a smart meter node owned and set up by the distribution system owner (DSO) to monitor the prosumer's energy input and output. The energy tokens purchased by the prosumer can be reclaimed as energy from the DSO through a prepaid smart contract system, where the prosumer pays for the desired energy in advance by depositing the appropriate number of tokens. The smart meter assigned to the prosumer sends its consumption data to the smart contract; its subscription balance is updated accordingly until it is exhausted. In case of early cancellation, the prosumer receives his change back from the smart contract. This system requires the use of a bidirectional smart meter that measures energy in two directions: how much energy is drawn from the grid and how much excess energy flows back into the grid. The flow of electricity into and out of the grid is measured by a bidirectional meter. A bidirectional meter differs from a conventional meter in that it displays three different readings: (1) Delivered, (2) Received, and (3) Net, whereas the conventional meter displays only one value: (1) Delivered. Delivered is the electricity coming in, received is the electricity going out, and net is the difference between the two values. A smart contract that conforms to the ERC20 standard manages the total supply, transfers, and account balances of all ERC20 tokens. An ERC20 contract establishes a set of rules that all fungible Ethereum tokens must follow. Each ERC20 smart contract must implement these features:

- totalSupply: A method that defines the total supply of tokens. When this limit is reached, the smart contract refuses to create new tokens.
- balanceOf: A method that returns the number of tokens that a given account has.
- transfer: A method that transfers a specified number of tokens from the total balance to a specified account.
- transferFrom: A transfer method that transfers tokens between user accounts.
- approved: This method checks whether a smart contract is allowed to allocate a certain number of tokens to a user, taking into account the total supply.
- allowance: This method is exactly the same as approved, except that it checks whether a user has enough funds to send a certain number of tokens to another.

Another function is added for smart meter nodes to claim a token when the required energy for the application is injected into the utility grid. The token is automatically granted when it is confirmed that the function caller is the smart meter node. Algorithm 1 uses pseudocode to describe this function. As shown in Algorithm 1, this function requires the sender to specify a PoL as a parameter. This PoL must be granted by a specific location verification smart contract, through a process that is described in detail in this document. Each PoL is identified by a unique Id that can only be incremented internally when a new PoL is requested. Each computed PoL is associated with a structure that has a position and

a unique Id. For each destination node, the most recently computed PoL is stored in the ledger and is associated with its address by a mapping that can be accessed via a getter specified in the smart contract for location verification. Similarly, each time the ClaimToken function is called, a Request Id associated with the function caller's address is incremented. Because a PoL is required for each ClaimToken, the PoL Request Id and the ClaimToken Request Id for a given account address must be identical. Figure 1 illustrates the blockchain energy trading system described.

Algorithm 1 ERC20 Smart Contract: ClaimToken

```
Adrress[]: SmartMeterNodes
Structure: Position
    Uint x
    Uint y
End Structure:
Structure: Pol
    Uint RequestId
    Position Position
End Structure:
Mapping: Address → Uint: RequestId
Procedure ClaimToken(PoL P)
    If msg.sender ∉ SmartMeterNodes Then
        Revert()
    End If
    If verify(Pol != True)
        Revert()
    Else
        Transfer(msg.sender, 1)
        RequestId(msg.sender) ← RequestId(msg.sender) + 1
    End If
End Procedure
```

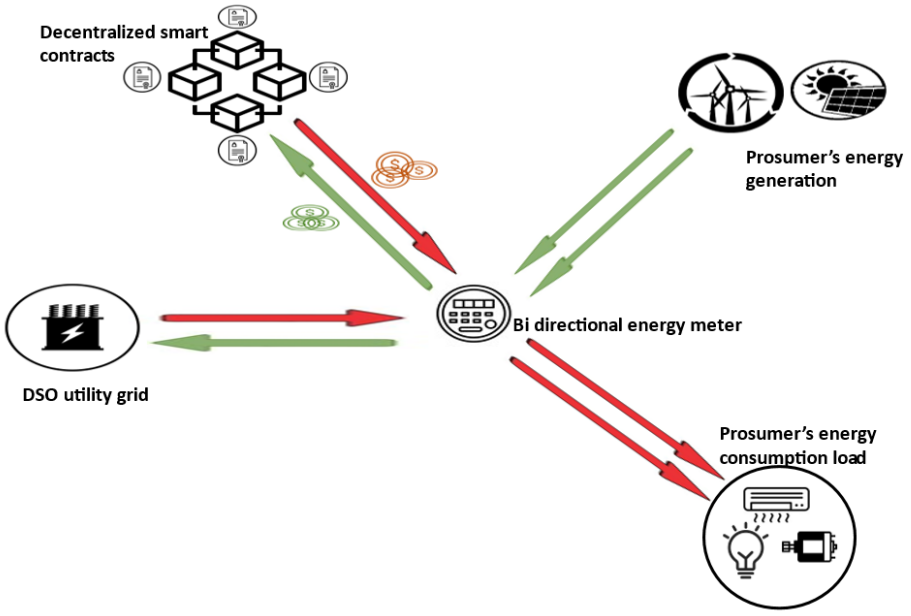

Figure 1. Blockchain enabled P2P trading scheme.

4. Proof of Location for Blockchain Enabled P2P Trading Scheme

PoL is significant in all applications when services can be claimed according to the origin of the request location. This is the case in blockchain-based energy applications, as tokens are delivered only if requested by the corresponding certified smart meter deployed

at a specific fixed and known location. The PoL scheme is detailed in full in this subsection. First, all entities involved in the system architecture are introduced. Next, the model architecture and the PoL setting procedure are explained. Finally, we describe how all the security vulnerabilities in setting a reliable PoL are closed by the threat and trust model of the proposed design.

4.1. Entities

In the proposed system architecture, there are four distinct entities:

- The target nodes: These are the nodes that need their locations confirmed before their transactions may be recorded in the blockchain's public ledger. In the context of the proposed P2P energy trading system, the target nodes are prosumer nodes, which are set and certified to request tokens upon the appropriate energy aggregation to the grid.
- Location prover nodes: These are end nodes that compete for a financial reward by calculating the location of a target node. To be a potential candidate for the location proving process, prover nodes must stake a certain amount of tokens in a specified smart contract. If a prover's claimed location turns out to be false following verification, the prover forfeits his or her wager. If, following verification, the claim is found to be correct, the prover is rewarded with tokens proportionate to the amount of money wagered. This turns the target location claim into a Token Curated Registry in which curators (prover nodes) bet financially on the validity of their claim (target node's location).
- The verifier: This is the entity in charge of verifying the target node's location, which is provided by the assigned group of provers. The verification process is performed by an intelligent independent entity, i.e., a smart contract, which is tasked with rewarding trustworthy LPs and penalizing fraudulent ones in the proposed system. A smart contract's verification ensures that it is decentralized and free of bias, collusion, or corruption. The prover nodes in the proposed scheme use the Difference Time of Arrival (TDoA) algorithm to calculate the distance between them and the target node, rather than the specific location of the target node. Trilateration, which is implemented in the location verifier smart device, is used to determine the exact location.
- Sink node: It collects and transmits to the location verifier smart contract the distances estimated by the selected set of prover nodes that separate them from the target node in question. The sink node is used to transfer locations calculated by different provers in a single transaction, rather than many transactions.

4.2. PoL Primary Model Architecture

4.2.1. Definition of Geographic Ranges for Prover and Target Nodes

As the location of the target node is determined by trilateration based on the time difference of the arrival of a transmitted RF signal, both the target and its assigned prover nodes must be in close proximity to ensure the accuracy of the calculated position. For this purpose, appropriate geographic square zones are defined that cover the entire area of interest and have no overlap. The geographic areas, as well as the coordinates where the target nodes should be located, are defined in the location verifier smart contract. Each target node i, denoted TN_i, is an object consisting of: a location Li represented by two coordinates (x_i, y_i), and a particular geographic area Gi, where: $\forall x, y \in [a, b]$, $G(x, y) = x \times y$, where a and b are the area boundaries, and a is a set of potential prover nodes and selected prover nodes. Each node can apply to be an anchor in a permission-free manner by submitting its exact location coordinates to the location verifier smart contract and is accordingly connected to the corresponding target nodes via a mapping. Even if the location of a particular target node in the verifier smart contract is calculated relative to the provers' claimed position, prover nodes in the proposed system cannot afford to lie about their position and would have to constantly update it in the location verifier smart contract.

Any misstatement of position would adversely affect to the prover in question. This will become clearer as the paper progresses.

4.2.2. Location Verification Process

The target nodes of the proposed P2P energy trading system are smart meter nodes configured to monitor prosumers' energy contributions to the power grid, requesting tokens when the associated prosumer delivers the corresponding energy. Because the private key needed to sign transactions is contained in the source code running on these nodes, anyone who gains access to the node's source code can access the private key. This makes the system vulnerable to identity theft, especially by the consumer to whom the smart meter is assigned. Consequently, transactions generated by the aforementioned target nodes must include a PoL to interact with the smart contract responsible for granting tokens to prosumers in accordance to their energy aggregation. Although location is not an absolute proof of identity, a PoL provides more security alternatives to the system.

Once a suspected fraudster linked with a particular target node is identified, proof of its dishonest act can be found in the public ledger, which contains transactions associated with the transmission of data to the blockchain that is incompatible with the smart meter readings in question. A target node sends a request to the location verifier smart contract before sending the aforementioned transactions to the corresponding smart contract. The location verifier smart contract then sends an event to notify the potential registered prover nodes in the same area as the target node in question. To be considered for participation in the target node's location verification process and potentially obtain a financial reward, interested prover nodes must make a deposit of at least a specific threshold. It is proposed to set a time limit for the submission of an application for a prover node and to stop accepting applications after the time limit expires. In such an application, a timer function with a specified time limit and a callback function are usually used. The callback function is automatically called when the selected time period ends, and the timer is triggered each time it is called. However, in most cases, implementing timers in a smart contract requires the use of an oracle, which means that the timers are implemented off-chain. The function to trigger the smart contract's timer is implemented on-chain. When the set time expires, an event is triggered to notify the associated off-chain timer application, which in turn invokes the callback function in the smart contract.

However, as there is no callback function in our context, the desired on-chain timer application is applicable. When a target node makes a new request for a location trace, the timer starts counting. This also triggers an event that notifies registered provers in the same geographic area that they can apply to be PoL verifier nodes for the target node's specific request. As the smart contract may receive overlapping or concurrent PoL requests from different target nodes, each target node has its own assigned timer. New prover applications are not accepted after the timer expires. The pseudocodes of Algorithm 2 and the flowchart in Figure 2 show how this is implemented in the location verifier smart contract.

The random selection of provers among the candidates begins whenever a new prover application is denied.

Algorithm 2 Location Verifier Smart Contract: RequestPoL, Apply/Deposit

Address: TargetNode
Uint: DepositThreshold
Uint: TimerDeadLine
Mapping: Address → **Address[]**: TargetNodePotenialProvers
Mapping: Address → **Uint:** PoLRequestId
Mapping: Address→ **Address[]**: TargetNodeCandidateProvers
Mapping: Address→ **Uint**: ProverDepositBalance
Mapping: Address →**Uint:** Timer
Procedure RequestPoL
    Timer(msg.sender) ← Time.Now()
    PoL(msg.sender) ← Time.Now()
    emit event: ProversApplicationOpen
End Procedure
Procedure Payable Apply/Deposit(TargetNode)
    Timer ← Timer(TargetNode)
    If msg.value ≥ DepositThreshold **Then**
        Revert()
    End If
    If Time.now() − Timer ≤ TimerDeadLine **Then**
        TargetNodeCandidateProvers(TargetNode).push(msg.sender)
        ProverDepositBalance(msg.sender) ← msg.value
    Else
        Emit Event: NotifyTargetNode(TargetNodeCandidateProvers)
    End If
End Procedure

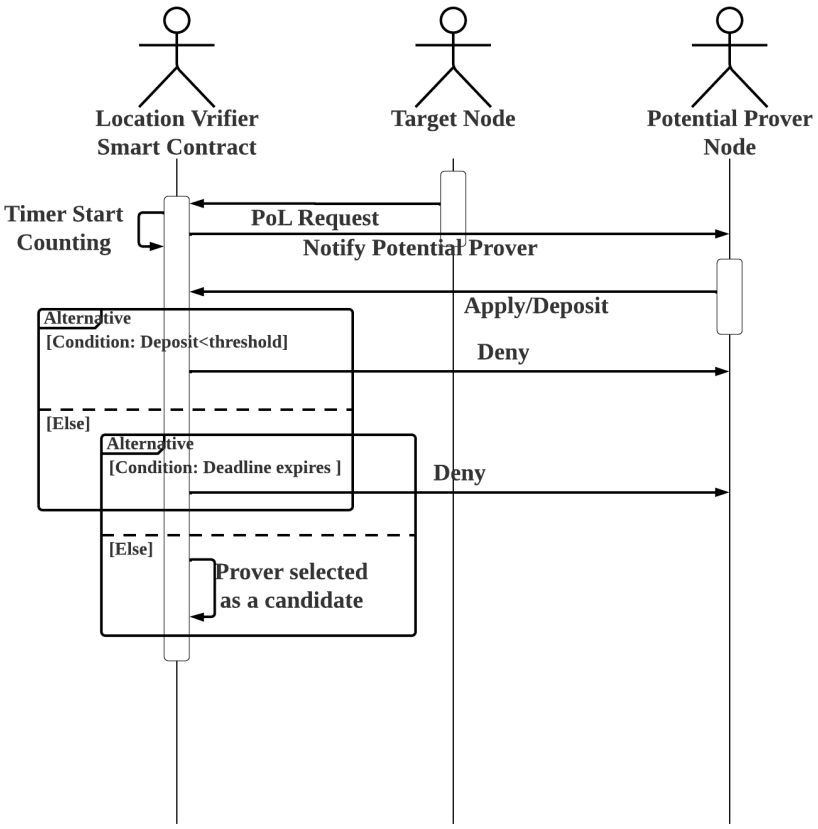

Figure 2. Flowchart illustrating the process by which potential prover nodes apply for location verification for a given PoL request from the target node.

4.2.3. Provers Random Selection

The random selection of the provers reduces the likelihood of collusion by reducing the probability that verifiers know each other. However, it must be based on an unpredictable random number generated by the location verifier smart contract. The generation of a random number by a smart contract, in contrast, is less obvious. There are only three approaches for generating random numbers with a smart contract. The first approach is to use the current block hash, block hash size, or current difficulty, which is determined by the block mining throughput. However, miners have a high degree of control over all of these parameters. In fact, it is the miners who calculate the block hash, and they might collude to mine blocks with a desired throughput, making this approach risky and the generated random number predictable. The second option is to have the random number generated outside of the smart contract by an oracle. This approach, however, is purely centralized and goes against the spirit of decentralization of the blockchain. It keeps a loophole open for bribed, corrupt central entities, especially if the generation of the random number has a financial incentive, as in our case. The last and most popular option is to use oracle-based verifiable random functions (VRF). The argument of the VRF oracle is that it cannot predict when the smart contract in question will request a random number. Therefore, it uses the timestamp of the request to generate a random number using an unpredictable but verifiable random function, which it then sends to the smart contract along with the proof of random generation. The VRF is made up of three different functions. They are, respectively: $Generate(x) = (P^{Key}, S^{Key}); F(x, S^{Key}) = y; PROOF(y, P^{Key})$. The VRF must satisfy three conditions, namely:

- Uniquenes: $\exists\ (x_1, y_1), (x_2, y_2) : PROOF(y_1, P^{Key1}) = PROOF(y_2, P^{Key2})$.
- Pseudo-randomness: Given a set of n inputs $S_{input} = x_1, \ldots, x_n$ and their respective set of output $S_{output} = y_1, \ldots, y_n$, there is no pattern linking outputs together.
- Provability:

The first function generates a Public/Secret key P^{Key} and S^{Key} based on the seed input x. The second function generates a random output from the seed input and S^{Key}; the third function is a proof of the correctness of $F(x, S^{Key})$, which can be verified using the output y and P^{Key}. Although VRF is a proven random number generation method in today's smart contracts [31,32], it is not appropriate for our application because it also relies on a smart contract-generated seed that is not random and can be manipulated. Although the three basic random number generation methods did not seem suitable for our proposed scheme, a thorough review of the literature led us to a model that did. The authors in [33] presented a roulette game implemented with a smart contract. Players wager a certain number of tokens on a number and win if their chosen number comes up in the smart contract's lucky draw. A game owner sells the tokens to the players and takes them back if the players lose, similar to casino roulette.

As the game owner and the player are counter-parties who would never collude, the game owner selects a random number, encrypts it, and sends it first to the smart contract in charge of the draw; after both the game owner's and the player's numbers are committed to the blockchain's public ledger, the game owner decrypts the number. Using the owner's public key, the smart contract checks whether the revealed number matches the encrypted number. If it does, it generates a random number using both the player and owner numbers. The owner starts first to ensure that he cannot manipulate the final result, and neither can the player, as the owner's number is revealed only after the player has committed his number to the blockchain. This scheme ensures fully decentralized random number generation in an application with counter-parties (cannot collude). The DSO is the entity in the proposed architecture for which the exact locations of the target nodes are important.

As the DSO provides the tokens and grants them to the target nodes through energy aggregation, other nodes might be tempted to falsify the computed location to obtain the tokens without satisfying the agreed conditions. Because the tokens are used to pay the DSO for power consumption, the reliability of the system is of paramount importance to the DSO. Thus, the proposed system is similar to a roulette game where the DSO is the

owner of the game and the prover nodes are the players. For this reason, we chose to use the same random number generation as implemented in [33]. Having its public key stored in the smart contract, the DSO starts the process by sending an encrypted number of its choice. Only after the DSO transaction is committed to the blockchain ledger can the prover node candidates send their own number to the blockchain, and the prover node candidates agree on the sink node that transmits their picked number with their respective signature. However, because identification in a permissionless blockchain environment is pseudo-anonymous, DSO can introduce covert nodes that use unknown public keys to manipulate the numerical sequence of prover candidates. For this reason, the sink node responsible for transmitting the prover number sequence should be an identified non-DSO node, such as a registered prosumer. The sink node would be the last prover to add a number to the sequence of numbers transmitted to the smart contract. After the numbers are committed to the blockchain and the corresponding signatures are verified, the DSO reveals its number, which is verified by the smart contract. If it matches the encrypted message, both the DSO and prover numbers are used by the smart contract to randomize the prover. See Figure 3 for how the encrypted DSO number is revealed and verified.

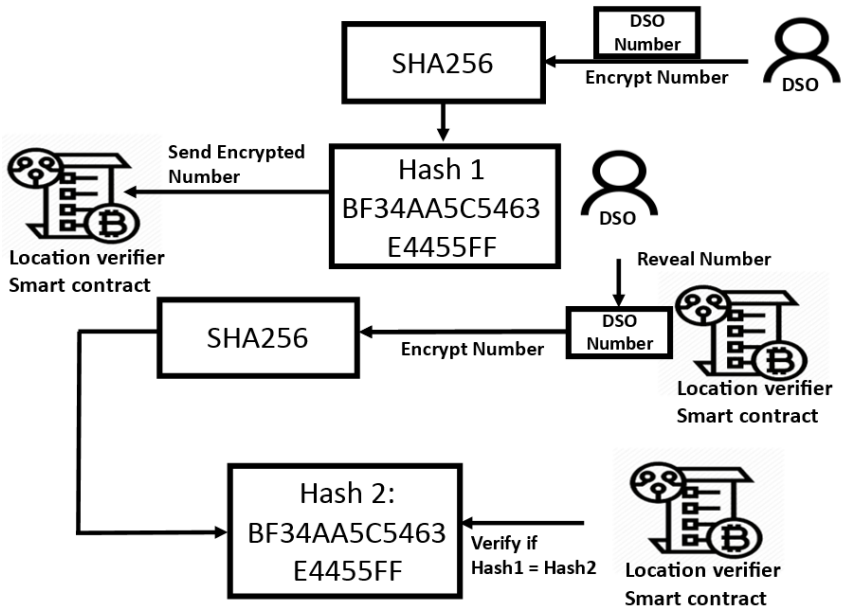

Figure 3. Process by which the DSO sends an encrypted number, then reveals it, and how it is verified by the location verifier smart contract.

The functions of the smart contract involved in random number generation are explained using the pseudocode of Algorithm 3 and the flowchart in Figure 4.

Random number generation is used for random selection of the prover. The function in question is described in pseudocode in Algorithm 4.

Algorithm 3 Location Verifier Smart Contract: SendDsoEncryptedNumber, SendProversNumber, DsoRevealNumber

```
Address: DSO
Mapping: Address → Uint: DsoSeedNumber
Mapping: Address → Uint: NonDsoSeedNumber
Mapping: Address → Bytes32: EncryptedDsoNumber
Procedure DsoSendEncryptedNumber(Bytes32 EncryptedNumber, Address TargetNode)
    EncryptedDsoNumber(TargetNode)← EncryptedNumber
    Bool DsoNumberCommitted← True
    emit event: NotifyProversToSendTheirNumber
End Procedure
Procedure SendProversNumber(Uint Number, Address TargetNode)
    If DsoNumberCommitted = True Then
        NonDsoSeedNumber(TargetNode)← Number
        Bool ProversNumberCommitted← True
        Emit Event: NotifyDsoToRevealNumber
    Else
        Revert()
    End If
End Procedure
Procedure DsoRevealNumber(Uint Number, Address TargetNode)
    If ProversNumberCommitted = True Then
        EncryptedDsoNumber ← EncryptedDsoNumber(TargetNode)
        c ← keccak256(abi.encode(Number))
        If c = EncryptedDsoNumber then
            DsoSeedNumber(TargetNode)← Number
        Else:
            Revert( )
        End If
    Else
        Revert()
    End If
End Procedure
```

4.2.4. Target Node Position Determination

The location verifier contract notifies the target node of its assigned prover after the prover nodes have been assigned. The target node, in turn, sends a RF message to each prover assigned to it, and each prover replies with the distance separating it from the target node calculated using TDoA. Before sending it back to the target node, each prover node signs the distance it has calculated. The location verifier smart contract verifies the signed distances received by the target node from the provers. Trilateration is used in TDoA localization to calculate the position of the target node based on the distances between the target node and at least three anchor nodes with known positions. However, three distances from three separate anchor nodes are sufficient to position the target node by solving the formula for the set of three equations, given a target node with coordinates (x_T, y_T) and three anchor nodes, where the coordinates of each anchor i are (x_i, y_i) and d_i is the distance between the target node and each anchor i.

$$\begin{cases} d_1^2 = (x_T - x_1)^2 + (y_T - y_1)^2 \\ d_2^2 = (x_T - x_2)^2 + (y_T - y_2)^2 \\ d_3^2 = (x_T - x_3)^2 + (y_T - y_3)^2 \end{cases} \quad (1)$$

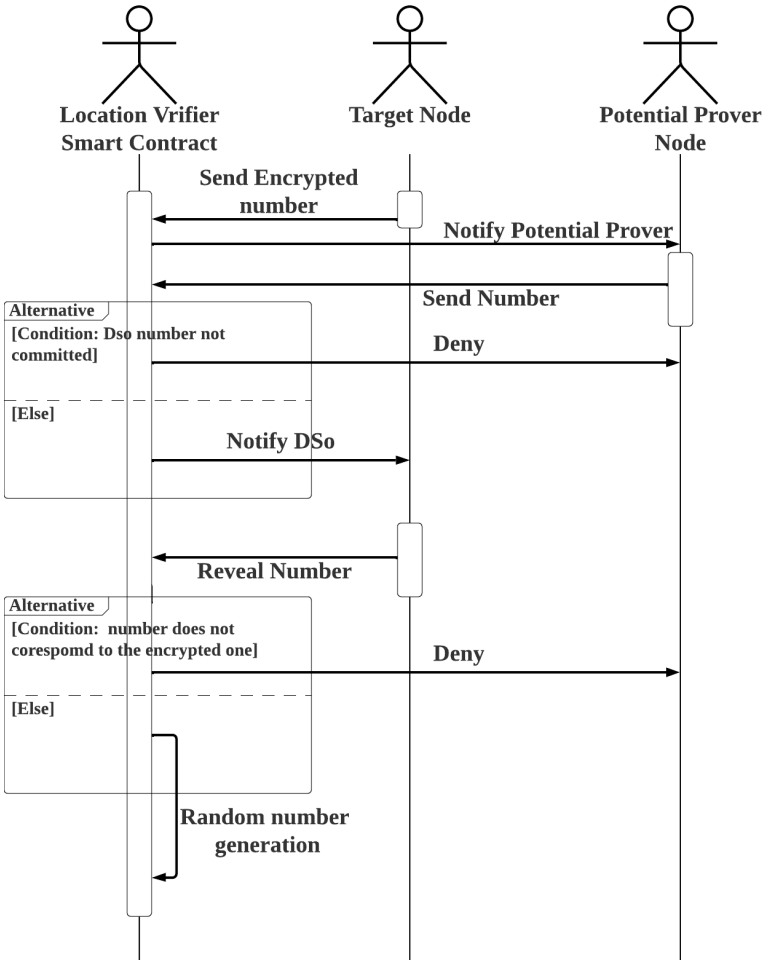

Figure 4. Random number generation process by the location verifier smart contract.

Algorithm 4 Location Verifier Smart Contract: SelectProvers

Mapping: Address → **Address[]**: TargetNodeCandidateProvers
Mapping: Address → **Address[]**: TargetNodeSelectedProvers
Mapping: Address → **Uint:** DsoSeedNumber
Mapping: Address ← **Uint:** NonDsoSeedNumber
Procedure SelectProvers(**Address** TargetNode)
  Uint: DsoNumber ← DsoSeedNumber(TargetNode)
  Uint: ProversNumber ← NonDsoSeedNumber(TargetNode)
  $i \leftarrow 1$
  while $i \leq$ TotalNumberOfSelectedProvers
    $k \leftarrow$ keccak256(abi.encode(DsoNumber, ProversNumber, i)) % 6
    Prover ← TargetNodeCandidateProvers(TargetNode)[k]
    if Prover $\notin$ TargetNodeSelectedProvers(TargetNode) **then**
      TargetNodeSelectedProvers(TargetNode).push(Prover)
    End If
    $i \rightarrow i + 1$
  End While
  Emit Event: NotifyTargetNodeOfItsSelectedProvers(TargetNodeSelected-textProvers(TargetNode))
End Procedure

As there are six selected anchor nodes, the target positioning can be done twice. After the six calculated distances are sent to the smart contract for location verification, the smart contract randomly selects two groups of three anchors to derive the target position by

trilateration using the two randomly selected anchor groups. The DSO and the prover nodes are involved in the random selection based on the same random number generation discussed earlier. Following the pseudocode described in Algorithm 5, the two positions calculated using the selected anchor node groups are determined.

Algorithm 5 Location Verifier Smart Contract: ComputetTargetNodePosition

```
1: Structure: Position
2:    Uint: x1
3:    Uint: y
4: End Structure
5: Structure: ComputedDistance
6:    Address: Prover
7:    Uint: d
8: End Structure
9: Structure: PositionEquationParameters
10:   Uint: AnchorNodeXPosition
11:   Uint: AnchorNodeYPosition
12:   Uint: Distance
13: End Structure
14: Mapping: Address → Position: NodePosition
15: Mapping: Address → ComputedDistance[6] TargetNodeComputed-
    Distances
16: Procedure ComputeTargetNodePosisition(Address TargetNode) Returns
    Position P1, P2
17:   PositionEquationParameters DistanceEquation
18:   PositionEquationParameters[2][3]  DistanceEquations
19:   Uint[]: AlreadyPicked
20:   i ← 1
21:        nonce ← 1
22:   While i ≤ 2
23:     j ← 1
24:     While j ≤ 3
25:       k← keccak256(abi.encode(DsoNumber2, ProversNumber2, nonce))% 6
26:       If k ∉ AlreadyPicked
27:          AlreadyPicked.push(k)
28:          c ← TargetNodeComputedDistance(TargetNode)[k]
29:          DistanceEquation.Distance← c.d
30:          x ← NodePosition(c.Prover).x
31:          DistanceEquation.TargetNodeXPosition← x
32:         y ← NodePosition(c.Prover).y
33:          DistanceEquation.TargetNodeYPosition→ y
34:          j ← j + 1
35:          nonce ← nonce+1
36:          DistanceEquations[i − 1][j − 1].push(DistanceEquation)
37:       Else
38:          nonce ← nonce+1
39:       End If
40:     End While
41:          i← i + 1
42:   End While
43:   Position P1, P2
44:   P1 → slove(DistanceEquations[0])
45:   P2 → slove(DistanceEquations[1])
46:   Return P1,P2
47: End Procedure
```

Provers are rewarded in proportion to their stake if the two calculated points match; otherwise, they lose their stake. The calculated position of the target node is then compared to its legitimate position. The target node is an identity thief if the two positions do not match; otherwise, the transaction is accepted. Assume that the six selected provers have no prior knowledge of each other and do not collude, and each computes the distance between itself and the target node. If one of the nodes delivers an incorrect distance, the two locations estimated by the smart contract verifier will have different coordinates, causing all verifiers to lose their bet. Therefore, if the prover wants to participate in location verification, the specified coordinates of the prover nodes in the smart contract must be correct and they must invest in the implementation of the agreed network communication protocol and media standard required for TDoA positioning. However, regardless of the anchor-node combination used, trilateration positioning will always result in the same derived point if all six prover nodes decide to collude and agree on a specific target position, and if they all send the distance that separates them from the agreed incorrect target position. As a result, this scheme is ineffective against prover node collusion. In the next section, we will look at how game theory is used to effectively address challenges. To solve the three expressions of Equation (1), we can expand the squares in each expression as follows:

$$\begin{cases} x_T^2 - 2x_1x_T + x_1^2 + y_T^2 - 2y_1y_T + y_1^2 = d_1^2 \\ x_T^2 - 2x_2x_T + x_2^2 + y_T^2 - 2y_2y_T + y_2^2 = d_2^2 \\ x_T^2 - 2x_3x_T + x_3^2 + y_T^2 - 2y_3y_T + y_3^2 = d_3^2 \end{cases} \tag{2}$$

The second expression is then subtracted from the first, and the third expression is subtracted from the second in Equation (2), yielding a system of two equations with two unknowns of the type:

$$\begin{cases} Ax_T + By_T = C \\ Dx_T + Ey_T = F \end{cases} \tag{3}$$

whose solution is given in Equation (4):

$$\begin{cases} x_T = \frac{CE-FB}{EA-BD} \\ Y_T = \frac{CD-AF}{BD-AE} \end{cases} \tag{4}$$

where:

$$\begin{cases} A = -2x_1 + 2x_2 \\ B = -2y_1 + 2y_2 \\ C = d_1^2 - d_2^2 - x_1^2 + x_2^2 - y_1^2 + y_2^2 \\ D = -2x_2 + 2x_3 \\ E = -2y_2 + 2y_3 \\ F = d_2^2 - d_3^2 - x_2^2 + x_3^2 - y_2^2 + y_3^2 \end{cases} \tag{5}$$

4.3. Enhanced Anti Collusion PoL Using Game Theory

In blockchain-based energy scenarios, there are two parties with two conflicting goals. The DSO might be tempted to misrepresent a location by claiming that the request did not originate from the smart meter node location in order to deny the prosumer a legitimate token claim. The prosumer, in turn, might be tempted to falsely claim that its request originated from the smart meter location in order to illegitimately claim an energy token. This leads to a possible game-theoretic scenario, explained below. Suppose a target node Tn with coordinates (x_t, y_t) requests PoL, where its legitimate position as deployed and set by the DSO is (x^*, y^*), and the position of the destination node calculated by TDoA trilateration is (x_c, y_c). In the proposed system, the involved parties can be classified into three categories:

- The parties for whom it is harmful if $(x_t, y_t) \neq (x^*, y^*)$, but beneficial for them if $(x_c, y_c) \neq (x^*, y^*)$, i.e., the DSO or other partners, because in the latter situation they could reject a legitimate prosumer's request for a token and enjoy the free energy

aggregation. However, if $(x_t, y_t) \neq (x^*, y^*)$, and $(x_c, y_c) = (x^*, y^*)$, they could deliver an energy token to a non-deserving node.
- Parties that are harmed when $(x_c, y_c) \neq (x_t, y_t)$ and $(x_t, y_t) = (x^*, y^*)$; these are the prosumer nodes and their partners that might be tempted to maliciously seek the position (x^*, y^*), to obtain a free energy token without having to supply the energy countervalue.
- Neutral parties who participate only for the financial reward.

Because the system is permissionless, all parties can apply to be location provers. It is proposed that N DSO prover nodes be defined in the smart contract, identified by their respective addresses. For prover applications, both DSO and non-DSO prover nodes are given fair slots, i.e., instead of a time-limited application, applications are open until a certain number m of applicants is reached, where $m/2$ applications are allowed for non-DSO nodes and $m/2$ for DSO nodes. Similar to the PoL scheme above, six examiners are randomly selected from the set of m, but there must be three DSO prover nodes and three non-DSO prover nodes. The function responsible for the random selection of examiners in this extended scheme is shown in the pseudocode of Algorithm 6. The distances asserted by the checkers must be sent to the smart contract in encrypted form. They can only be decrypted after they have all already been submitted to the blockchain.

Algorithm 6 SelectRandomProvers

```
Uint: NumberofProverApplicants
Mapping: Address → Uint: DsoSeedNumber
Mapping: Address → Uint: NonDsoSeedNumber
Address[NumberofProverApplicants/2]: NonDsoProverNodes
Mapping(NumberofProverApplicants/2: NonDsoProverNodes
Procedure ToggleBool(Bool b)
  If b = True Then
    b ← False
  Else
    b ← True
End Procedure
Procedure SelectRandomProvers(Address TargetNode)
  Bool DsoProverTurn← True
  i ← 1
  While i ≤ 6
    DsoNumber2← DsoSeedNumber(TargetNode)
    ProversNumber2← NonDsoSeedNumber(TargetNode)
    k← keccak256(abi.encode(DsoNumber2, ProversNumber2, i)) % (m/2)
    If DsoProverTurn = True Then
      Prover ← DsoProverNodes[k]
      If Prover ∉ TargetNodeSelectedProvers(TargetNode) then
        TargetNodeSelectedProvers(TargetNode).push(Prover)
      End If
      ToggleBool(DsoProverTurn)
    Else
      Prover ← NonDsoProverNodes[k]
      If Prover∉ TargetNodeSelectedProvers(TargetNode) then
        TargetNodeSelectedProvers(TargetNode).push(Prover)
      End If
      ToggleBool(DsoProverTurn)
    End If
    i ← i + 1
  End While
End Procedure
```

After the six encrypted distances are committed, they are decrypted and output by the respective prover nodes. Two target position points are derived by the location verifier smart contract. One is calculated using the three distances given by the DSO prover nodes, and the second is calculated using the distances given by the non-DSO prover nodes, according to the pseudocode in Algorithm 7.

Algorithm 7 Location Verifier Smart Contract: ComputeTargetNodePosition

```
Structure: Position
   Uint: x
   Uint: y
End Structure
Structure: PositionEquationParameters
   Uint: AnchorNodeXPosition
   Uint: AnchorNodeYPosition
   Uint: Distance
End Structure
Mapping: Address → Position: NodePosition
Mapping: Address → Uint[6]: TargetNodeComputedDistances /*The 3
first distances correspends to distances computed by DSO anchors, the rest by Non
DSO anchors*/
Procedure ComputeTargetNodePosisition(Address TargetNode)
 Returns Position P1, P2
   PositionEquationParameters DistanceEquation
   PositionEquationParameters[2][3] DistanceEquations
   i ← 1
   While i ≤ 2
     j ← 1
     While j ≤ 3
        d ← TargetNodeComputedDistances(TargetNode)[j − 1]
        DistanceEquation.Distance← d
        Prover ← TargetNodeSelectedProvers(TargetNode)[j − 1]
        x ← NodePosition(Prover)
        DistanceEquation.TargetNodeXPosition← x
        y ← NodePosition(Prover).y
        DistanceEquation.TargetNodeYPosition← y
        DistanceEquations[i−1][j−1].push(DistanceEquation)
       j ← j + 1
      End While
   i ← i + 1
   End While
   Position P1, P2
   P1 → slove(DistanceEquations[0])
   P2 → slove(DistanceEquations[1])
   Return P1,P2
End Procedure
```

Note that the only case in which DSO prover nodes can be tempted to lie is when the target coordinates $(x_c, y_c) \neq (x^*, y^*)$, if the target coordinates claimed by the DSO nodes $(x_c, y_c) = (x^*, y^*)$, this must be true, whereas non-DSO prover nodes can only be tempted to lie about the position of the target node by claiming $(x_c, y_c) = (x^*, y^*)$. If the target coordinates computed by the non-DSO nodes are $(x_c, y_c) \neq (x^*, y^*)$, this must be true. This allows game theory to be applied to this system, as shown in Figure 5.

We assume that the target positions calculated by the smart contract based on the distances of the DSO and non-DSO providers are (x_d, y_d) and (x_p, y_p), respectively. We can discern for both DSO and non-DSO prover nodes in which they Certainly Tell the Truth (CTT) or Certainty Lie (CL), or situations in which no judgement can be made but At Least

One Certainly Lies (ALOC). These situations are presented in Equations (6) and (7) for DSO and non-DSO prover nodes, respectively.

$$\text{DSO prover node:} \begin{cases} \text{CTT: } (x_d, y_d) = (x^*, y^*) || (x_d, y_d) = (x_p, y_p) \neq (x^*, y^*) \\ \text{CL: } (x_d, y_d) \neq (x_p, y_p) \& (x_p, y_p) \neq (x^*, y^*) \\ \text{ALOC: } (x_d, y_d) = (x^*, y^*) \& (x_p, y_p) \neq (x^*, y^*) \end{cases} \quad (6)$$

$$\text{Non-DSO prover node:} \begin{cases} \text{CTT: } (x_p, y_p) \neq (x^*, y^*) || (x_p, y_p) = (x_d, y_d) = (x^*, y^*) \\ \text{ALOC: } (x_p, y_p) \neq (x^*, y^*) \& (x_d, y_d) = (x^*, y^*) \end{cases} \quad (7)$$

Tables 1 and 2 illustrate gains for both DSO and non-DSO provers' according to their adopted game strategy (Lie/Tell the truth).

Table 1. Strategy played versus gain in the game theory scenario between DSO and non-DSO provers, where $(x_t, y_t) = (x^*, y^*)$.

Non-DSO Prover Node \ **DSO Prover Node**	**Lie:** $(x_c, y_c) \neq (x^*, y^*)$	**Truth:** $(x_c, y_c) = (x^*, y^*)$
Truth: $(x_c, y_c) = (x^*, y^*)$	$(-10, -10)$	(10, 10)
Lie (declared without computing): $(x_c, y_c) = (x^*, y^*)$	$(-10, -10)$	(10, 10)

Table 2. Strategy played versus gain in the game theory scenario between DSO and non-DSO provers, where $(x_t, y_t) \neq (x^*, y^*)$.

Non-DSO Prover Node \ **DSO Prover Node**	**Truth:** $(x_c, y_c) \neq (x^*, y^*)$ & $(x_c, y_c) = (x_t, y_t)$	**Lie:** $(x_c, y_c) \neq (x^*, y^*)$ &$(x_c, y_c) \neq (x_t, y_t)$
Lie: $(x_c, y_c) = (x^*, y^*)$	$(-10, -10)$	(10, 10)
Truth: $(x_c, y_c) \neq (x^*, y^*)$ &$(x_c, y_c) = (x_t, y_t)$	(20, 20)	(30, -30)

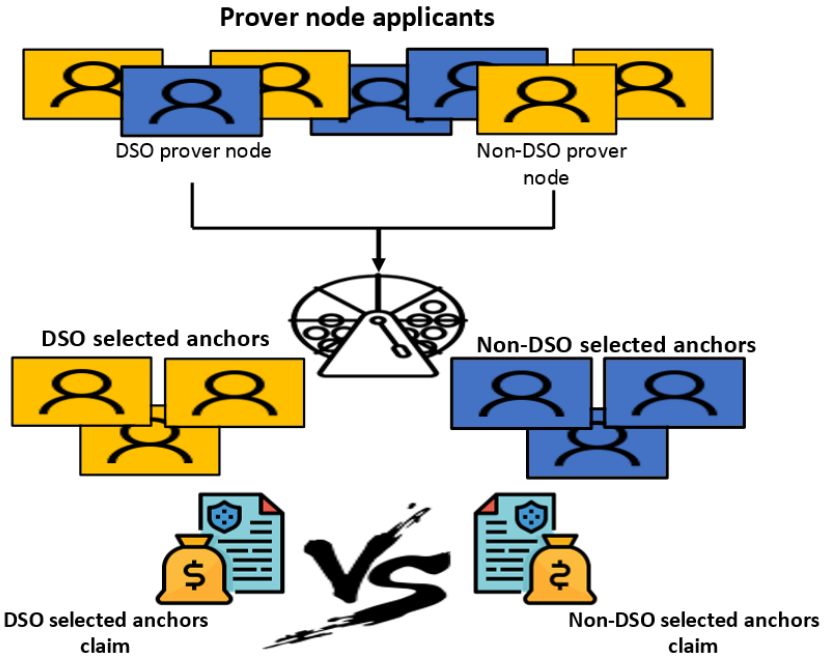

Figure 5. Game theory scenario in target node positioning between DSO and Non-DSO anchors.

To calculate the Nash equilibrium, we calculate the total gain for both entities in the two situations described in Tables 1 and 2—when the entity lies and when it is truthful. For

the non-DSO prover node, the total gain is 50 when it is truthful and −20 when it lies. For the DSO-prover node, the total gain is 30 if it is truthful and −60 if it lies. Although the non-DSO prover node has an advantage in this game, the Nash equilibrium for both parties is to be truthful. The proposed PoL procedure is secure and safe against prover-target collusion. As for target-target collusion, it is not a threat to the proposed P2P application, as any target-target collusion is irrelevant if it does not usurp the respective legitimate position of the smart meter.

After P1 and P2 are calculated by the location verifier smart contract corresponding to target position claimed by the DSO and non-DSO sets of anchors, respectively, the PoL for the particular target node request is set according to the pseudocode in Algorithm 8. After the PoL is computed, it must be included as a parameter in the *ClaimToken* function noted in Algorithm 1. To verify the validity of the PoL passed by the caller in the *ClaimToken* function, the ERC20 smart contract delivering the energy token checks the validity of the asserted PoL by first verifying that it was issued by the location verifier smart contract by calling a getter function that accesses the key-value mapping linking the destination node addresses to their most recently computed PoL. The latter must be identical to the one passed in the *ClaimToken* function call. The ERC20 smart contract that issues the energy tokens must also verify that the PoL Id matches that of the *ClaimToken* call. Both Ids are associated with the respective function caller by mapping. It should be noted that both Ids are constrained and can only be incremented when the appropriate function is called. Because there must be a PoL for each token claim, the two Ids must be identical. How the PoL is checked when a token is claimed is shown in the pseudocode in Algorithm 9.

Algorithm 8 Location Verifier Smart Contract: SetPoL, GetNodePol

```
Structure: PoL
    Uint: PoLRequestId
    Position: P
End Structure
Mapping: Address → PoL: TargetNodePoL
Procedure SetPol(PcomputedByDsoAnchors, PcomputedByNonDsoAnchors)
    PoL ProofOfLocation
    P1 ← PcomputedByDsoAnchors
    P2 ← PcomputedByNonDsoAnchors
    P* ← NodePosition(TargetNode)
    If P1=P2 Then
        ProofOfLocation.P← P1
        ProofOfLocation.PoLRequestId← RequestId(TargetNode)
    ElseIf P1 != P2 Then
        If P1 != P* & P2 != P* Then
            ProofOfLocation.P← P2
            ProofOfLocation.PoLRequestId← RequestId(TargetNode)
        Else
            Revert() /* The proof of location is not considered and need to be
    recomputed*/
        End If
    End If
End Procedure
Procedure GetNodePol(Address Addr) Returns(PoL)
    Return TargetNodePoL(Addr)
End Procedure
```

The entity diagram relationship in the proposed PoL scheme using the location verifier smart contract is described in Figure 6.

Algorithm 9 ERC20 Smart contract: Verify

```
Address: LocationVerifierSmartContractAddress
Mapping: Address → Position:  TargetNodeDefinedPosition
Procedure Verify(Pol L) Returns (Bool)
    c ← CallLocationVerifierSmartContract(LocationVerifierSmartContractAddress)
    l← c.GetNodePol(msg.sender)
    Id ← RequestId(msg.sender)
    N← TargetNodeDefinedPosition(msg.sender)
    if (L = l & l.RequestId = Id & L.Position = N) Then
        Return true
    Else
        Return false
    End If
End Procedure
```

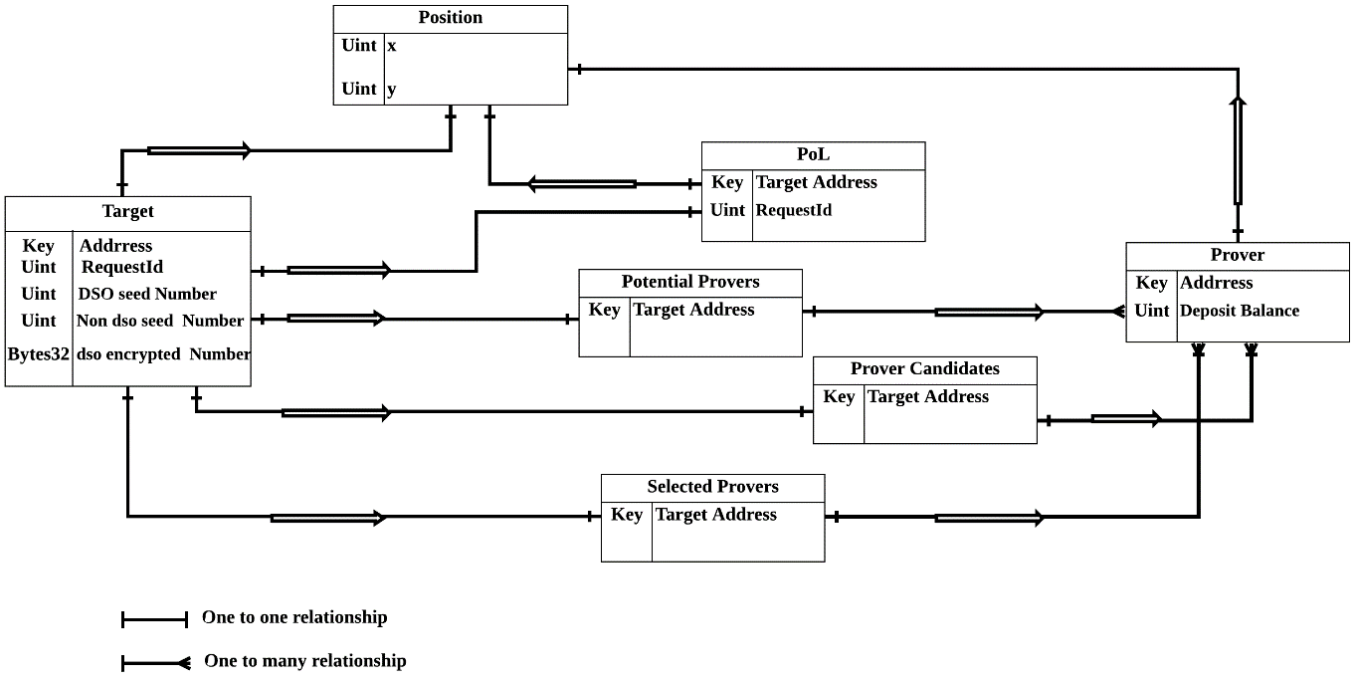

Figure 6. Entity relationship diagram of the location verifier smart contract.

Time Difference of Arrival

Time difference of arrival (TDoA), also known as multilateration, is a proven method for positioning RF transmitters. Using three or more anchor receivers, TDoA positions a signal source based on the difference in arrival times at the receivers. Before a round of TDoA positioning begins, both the target and its assigned anchor nodes must establish time synchronization. Here, $t_0 = 0$ refers to the exact time at which the target sends a RF message to the respective anchor nodes. The distance between the anchor and the RF sender anchor node is calculated based on the time it takes for the RF message to reach the anchor node, as given in Equation (8):

$$distance = SpeedOfLight \times time \tag{8}$$

The time delay is determined using the cross-correlation in the time domain between the received signals at two anchors, which peaks when $t = \tau$, where τ is the time delay between the two signals. Given a set of n received signals $R_i(t)$, each received at anchor i where $0 < i \leq n$; Given a pair of sampled received signals $R_1(n), R_2(n)$. Time delay at

which the two signals are received is computed by time domain cross-correlation and is found according to Equation (9):

$$\begin{cases} corr(m) = \sum_{m=0}^{N-1} R_1(m)R_2(m+n) \\ corr(\tau_{1,2}) = Max(\sum_{m=0}^{N-1} R_1(m)R_2(m+n)) \end{cases} \quad (9)$$

where $\tau_{1,2} = \tau_1 - \tau_2$

Thus: $c\tau 12 = c\tau 1 - c\tau 2 = d_1 - d_2$; where d_1 is the target distance from anchor 1, and d_2 is its distance from anchor 2. The hyperbola function of the distances between anchor nodes 1 and 2 and the target is given in Equation (10):

$$\begin{cases} d_1 = \sqrt{(x_T - x_1)^2 + (y_T - y_1)^2} \\ d_2 = \sqrt{(x_T - x_2)^2 + (y_T - y_2)^2} \end{cases} \quad (10)$$

However, d_1 and d_2 are both unknown, only $d_1 - d_2$ is known, which was derived according to Equation (11):

$$d_1 - d_2 = \sqrt{(x_T - x_1)^2 + (y_T - y_1)^2} - \sqrt{(x_T - x_2)^2 + (y_T - y_2)^2} \quad (11)$$

To solve (x_T, y_T), two more equations are needed. Thus, we need two more pairs of anchors and the time lag between their received signals. Therefore, this approach requires six anchor nodes to position the target node. However, there are other numerical methods that can be used to calculate the target position with fewer anchor nodes, such as in [34].

The TDoA positioning accuracy is not really affected by the distance between the target node and the anchors, as TDoA is the positioning technique implemented by GPS localization and uses satellite anchors that are thousands of kilometers away from the target nodes. TDoA accuracy is mainly influenced by the quality of correlation, which depends on several factors:

- The nature of the received RF signal (especially its bandwidth);
- The different characteristics of the receivers;
- The different propagation paths between the transmitter and the receivers;
- The correlation method used.

However, the range between anchors and targets must be consistent with the wireless communication protocol (WCP) used. Table 3 shows the range in meters of different WCPs that allow precise TDoA localization [35].

Table 3. Distance coverage for different wireless communication technologies.

Wifi	Bluetooth	LoRaWan	ZigBee
20–50 m	1–7 m	1000–2000 m	20–100 m

5. Experiment Results and Analysis

A private Ethereum blockchain was run on a machine with 4 cores CPU: Intel(R) Core (TM) i5-9300H CPU @ 2.40 GHz and a GPU: NVIDIA GeForce GTX 1650 with Max-Q design to evaluate the performance of the proposed platform. It is worth noting that the goal of the experiment is not to verify the functionality of the platform, as this has already been done during the development phase with Remix IDE. However, we decided to focus on evaluating the performance when we increase the number of requests to blockchain nodes and their sending rate, as well as the smart contract functions execution gas cost. Accordingly, blockchain nodes are created running on different ports and connected to each other, adding other nodes as peers. Each created node, in its entirety, represents a target node that requests PoL with a set of subscribed provers that apply to be location verifiers for the respective target node request. Using a node-js application and the web3 library, each blockchain node is configured to invoke a series of transactions. The flowchart

in Figure 2 shows the transaction sequence used, where transactions are not issued until the smart contract is notified of the corresponding event. The number of network nodes is specified in a genesis.json file, along with the Etash PoW mining difficulty. The selected difficulty was 0×04 in hexadecimal, which corresponds to an average mining time of 30 ms on the deployed machine. The metric evaluated is transaction latency, which is the time in seconds between the issuance of a transaction by the node invoking the smart contract and its delivery to the ledger. As can be seen in Figure 7, transaction latency increases as the sending rate increases. This is due to the block mining time and the fact that the block size on the Ethereum blockchain is limited to a maximum of 20 million gas, resulting in a limited number of transactions in a single block. The average transaction execution time is also quite high, which is due to the fact that some transactions in the sequence may not be in the same blocks. As some transactions in the sequence cannot be contained in the same blocks, the average transaction execution time is also quite high. In addition, the latency stabilizes after a certain transmission rate threshold, which is because the execution of the sequence is asynchronous and cannot exceed a certain pace. Figure 8 shows the gas consumption of each smart contract function. The high gas consumption of ComputeTargetNodePosition is noticeable here, as this function is very computationally intensive.

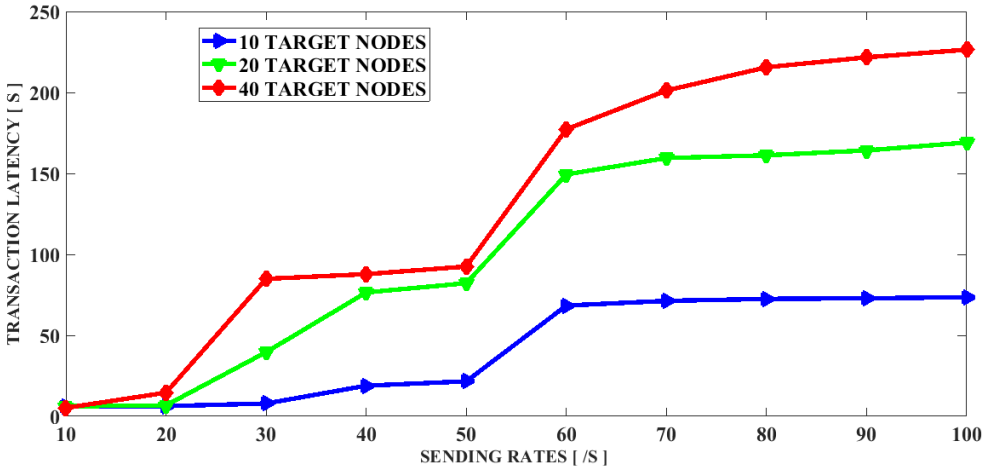

Figure 7. Transaction's latency with increasing send rates for different numbers of target node.

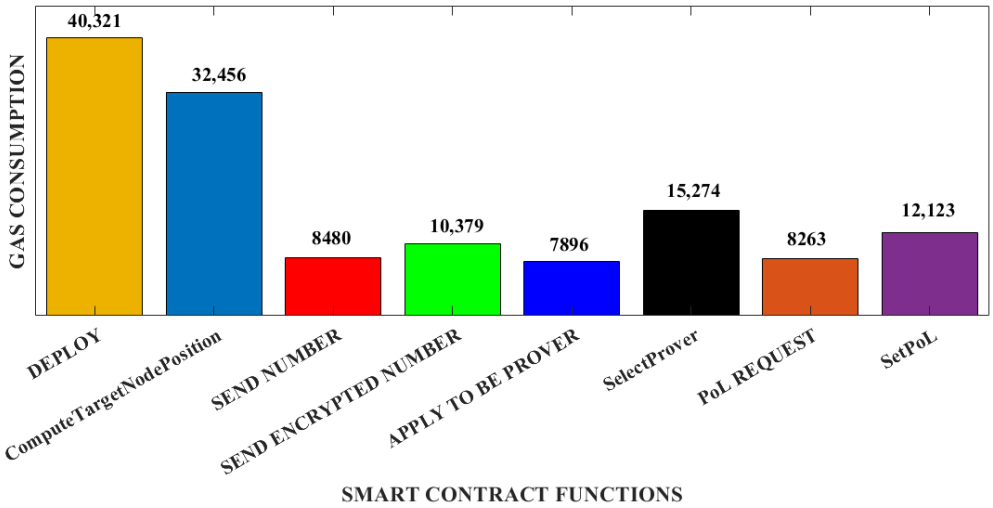

Figure 8. Location verifier smart contract gas consumption.

6. Conclusions

In this paper, we present a PoL for blockchain-based energy platforms. In the proposed scheme, no central entities are entrusted with the positioning of the target nodes ; rather,

the target locations are verified in a decentralized manner, which is enabled through smart contract random number generation and game theory suitable for the blockchain energy trading application. An Ethereum-based energy platform was presented, where prosumers collaborate to form a single provider supplying client consumers with a prepaid blockchain-based billing system. The potential and feasibility of such a platform was discussed, as well as how location in such a scheme can be verified in a decentralized yet reliable manner to be immune to potential fraud or collusion pertinent to blockchain energy trading systems. It has been shown how, with two counter-parties, as in the case of blockchain-based energy applications, a game-theoretic scenario arises in which equilibrium is only possible if all participating verifiers state the true origin of the token request, and how, with two counter-parties, it is possible to implement a decentralized and unpredictable random selection of verifiers to reduce the likelihood of collusion between them. As part of this work, a detailed description of the architecture and functioning of this platform and its implementation was undertaken. The goal of this work was to shed light on the relevance of the proposed platform and evaluate it using relevant metrics.

Author Contributions: Conceptualization, methodology, and writing, Y.M. and M.H.H.; software, Y.M.; validation, supervision, and revision, M.H.H., M.R.I., T.S.G. and M.M.; funding, M.H.H., M.R.I., T.S.G. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was conducted at the IoT and Wireless Communication Protocols Laboratory at the ECE department, International Islamic University Malaysia (IIUM). It is partially sponsored by the Malaysian Ministry of Higher Education (MoHE) Prototype Research Grant Scheme number PRGS19-0013-0057.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Acknowledgments: Yacine Merrad is grateful to IIUM Tuition Fee Waiver program.

Conflicts of Interest: The authors declare that they have no conflict of interest.

References

1. Antal, T.L.C.; Antal, M.; Mitrea, D.; Cioara, T.; Anghel, I. A Lockable ERC20 Token for Peer to Peer Energy Trading. *arXiv* **2021**, arXiv:2111.04467.
2. Munoz, M.F.; Zhang, K.; Amara, F. ZipZap: A Blockchain Solution for Local Energy Trading. *arXiv* **2022**, arXiv:2202.13450.
3. Buccafurri, F.; Lax, G.; Musarella, L.; Russo, A. An Ethereum-based solution for energy trading in smart grids. *Digit. Commun. Networks* **2021**. [CrossRef]
4. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System, Decentralized Business Review, 21260. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 15 April 2022).
5. Raval, S. *Decentralized Applications: Harnessing Bitcoin'S Blockchain Technology*; O'Reilly Media, Inc.: Newton, MA, USA, 2016, p. 118.
6. Aste, T.; Tasca, P.; Matteo, T.D. Blockchain technologies: The foreseeable impact on society and industry. *Computer* **2017**, *50*, 18–28. [CrossRef]
7. Nofer, M.; Gomber, P.; Hinz, O.; Schiereck, D. Blockchain. *Bus. Inf. Syst. Eng.* **2017**, *59*, 183–187. [CrossRef]
8. Biryukov, A.; Khovratovich, D.; Pustogarov, I. Deanonymisation of clients in Bitcoin P2P network. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; pp. 15–29.
9. Haber, S.; Stornetta, W.S. *How to Time-Stamp a Digital Document. Conference on the Theory and Application of Cryptography*; Springer: Berlin/Heidelberg, Germany, 1990; pp. 437–455 .
10. Brown, D.R.L. Recommended Elliptic Curve Domain Parameters. In *Standards for Efficient Cryptography*, 3rd ed.; Certicom Research: Mississauga, ON, Canada, 2010; p. 33.
11. Blundo, C.; Lovino, V.; Persiano, G. Private-key hidden vector encryption with key confidentiality. In Proceedings of the International Conference on Cryptology and Network Security, Kanazawa, Japan, 12–14 December 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 259–277.
12. Hoy, M.B. An introduction to the blockchain and its implications for libraries and medicine. *Med Ref. Serv. Q.* **2017**, *36*, 273–279. [CrossRef] [PubMed]
13. Coinmap. Available online: https://coinmap.org/ (accessed on 15 April 2022).
14. Luu, L.; Chu, D.-H.; Olickel, H.; Saxena, P.; Hobor, A. Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016; pp. 254–269.

15. Christidis, K.; Devetsikiotis, M. Blockchains and smart contracts for the internet of things. *IEEE Access* **2016**, 4, 2292–2303. [CrossRef]
16. Klomp, R.; Bracciali, A. On symbolic verification of Bitcoin's script language. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*; Springer: Cham, Switzerland, 2018; pp. 38–56.
17. Asuquo, P.; Cruickshank, H.; Morley, J.; Ogah, C.P.; Lei, A.; Hathal, W.; Bao, S.; Sun, Z. Security and Privacy in Location-Based Services for Vehicular and Mobile Communications: An Overview, Challenges, and Countermeasures. *IEEE Internet Things J.* **2018**, *5*, 4778–4802. [CrossRef]
18. Gartner, R.J.G.; Karimi, H.; Rizos, C. Applications of location–based services: A selected review. *J. Locat. Based Serv.* **2007**, *1*, 89–111.
19. Gupta, R.; Rao, U. An exploration to location based service and its privacy preserving techniques: A survey. *Wirel. Pers. Commun.* **2017**, *96*, 1973–2007. [CrossRef]
20. Saroiu, S.; Wolman, A. Enabling new mobile applications with location proofs. In Proceedings of ACM HotMobile, Santa Cruz, CA, USA, 23–24 February, 2009; pp. 1–6.
21. Zhou, Z.Z.L.; Zhao, X.; Wang, G.; Su, Y.; Metzger, M.; Zheng, H. On the Validity of Geosocial Mobility Traces. In Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets), Princeton, NJ, USA, 13–15 November 2013; pp. 1–7.
22. Pham, A.; Huguenin, K.; Bilogrevic, I.; Dacosta, I.; Hubau, J. Secure Run: Cheat–proof and private summaries for location–based activities. *IEEE Trans. Mob. Comput.* **2016**, *15*, 2109–2123. [CrossRef]
23. Javali, C.; Revadigar, G.; Rasmussen, K.; Hu, W.; Jha, S.; Alice, I.A. I Was in Wonderland: Secure Location Proof Generation and Verification Protocol. In Proceedings of the 2016 IEEE 41st Conference on Local Computer Networks (LCN), Dubai, UAE, 7–10 November 2016; pp. 477–485.
24. Zhu, Z.; Cao, G. APPLAUS: A privacy–preserving location proof updating system for location–based services. In Proceedings of IEEE INFOCOM, Shanghai, China, 10–15 April 2011; pp. 1889–1897.
25. Boureanu, I.; Mitrokotsa, A.; Vaudenay, S. Practical and provably secure distance bounding. *J. Comput. Secur.* **2015**, *23*, 229–257. [CrossRef]
26. Boureanu, I.; Vaudenay, S. Challenges in Distance Bounding. *IEEE Secur. Priv.* **2015**, *13*, 41-48. [CrossRef]
27. Gambs, S.; Killijian, M.; Roy, M.; Traore, M. PROPS: A privacy–preserving location proof system. In Proceedings of the IEEE 33rd International Symposium on Reliable Distributed Systems, Nara, Japan, 6–9 October 2014.
28. Kounadi, O.; Bernd, R.; Andreas, P. Privacy Threats and Protection Recommendations for the Use of Geosocial Network Data in Research. *Soc. Sci.* **2018**, *7*, 1–17. [CrossRef]
29. Nosouhi, M.R.; Yu, S.; Zhou, W.; Grobler, M. Blockchain for secure location verification. *J. Parallel Distrib. Comput.* **2020**, *136*, 40–51. [CrossRef]
30. Foamspace Corp. FOAM—The Consensus Driven Map of the World. 5 January 2018. Available online: https://foam.space/publicAssets/FOAM_Whitepaper.pdf (accessed on 7 April 2022).
31. Shu, F.; Lei, K. Vger: A VRF based cross-chain mechanism for blockchains. *J. Phys. Conf. Ser.* **2021**, *1780*, 012038. [CrossRef]
32. Emmanuel, M.; Chacko, A.N. *BSCDL: A Blockchain based Smart Contract Digitized Lottery Scheme*; EasyChair Preprint: Manchester, UK, 2020; p. 10. Available online: https://easychair.org/publications/preprint/W7Qp (accessed on 17 May 2022)
33. Du, M.; Chen, Q.; Liu, L.; Ma, X. A Blockchain-based Random Number Generation Algorithm and the Application in Blockchain Games. In Proceedings of the 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), Bari, Italy, 6–9 October 2019; pp. 3498–3503.
34. Phruksahiran, N.; Michanan, J. Iteration improvement of Taylor-series estimation using hyperbolic systems for FM-radio source localization in Bangkok. *Signal Image Video Process.* **2021**, *15*, 247–254. [CrossRef]
35. LoRa Alliance™ Strategy Committee, LoRaWAN Geolocation Whitepaper. January 2018. Available online: https://lora-alliance.org/sites/default/files/2018-04/geolocation_whitepaper.pdf (accessed on 7 April 2022).