Decentralized Byzantine Fault Tolerant Proof of Location

Dmitrijs Rjazanovs and Ernests Petersons

Department of Electronics and Telecommunications, Riga Technical University,
12 Azenes Street, LV-1048 Riga, Latvia

dmitrijs.rjazanovs@edu.rtu.lv

Abstract. The majority of modern location-based services rely on GPS to verify the location of different subjects. This approach is acceptable as long as the user is not interested in location data corruption, for instance, the usage of navigation utilities. On the other hand, when a service depends on the reliability of location data, then GPS is not acceptable. One of the most interesting examples of location data reliability dependent services is a smart contract, which needs to verify the location of a certain subject. The only way to verify a location within a smart contract is to rely on a third party. This fact breaks the decentralization principle of smart contracts. This paper introduces a novel algorithm of location proof within the smart contract by leveraging a p2p mesh network of Wi-Fi access points and a distributed key generation algorithm.

Keywords: Location Proof; Smart Contracts; Decentralization; P2P;

1 Introduction

The location-based services like package delivery in urban areas is something that everyone is using today. Internet-based technologies, such as e-commerce, made the acquisition of any goods almost flawless. A regular customer is not interested in the underlying processes of buying something online and having it delivered at their doorstep. This is because the customer has a payment card, a smartphone, and ten minutes of their time, and there is almost nothing to worry about. However, from the perspective of a business owner the organization of such service is complicated, tedious, expensive, and requires a lot of trust in other people. This includes at least, the development of a website, signing a contract with the bank acquirer and signing a contract with a logistics company. At least two of those three tasks are now obsolete, which are the bank contracts and logistics company.

To support the above statements, a survey of existing technologies will be provided here. In 2008, the probabilistic solution for “Byzantine Generals Problem” [1] for any

number of malicious nodes was published [2]. This allowed the implementation of a decentralized payment system called Bitcoin [2]. The theme of cryptocurrencies is beyond the scope of this paper and there are already a lot of investigations in this area. In 2013, the blockchain technology was extended [3] to support Turing-complete distributed code execution with one important property – unless the network is completely stopped, the code will be executed exactly as written and the state will be persistent and unforgeable (with quite a high probability). This invention got an informal name of “smart contracts" and the first such network was Ethereum. There are a lot of researches made on Ethereum and other related technologies, but we are not going to discuss the pros and cons of current implementation because this is beyond the scope of this paper.

One of the interesting cases of using smart contracts is robonomics. Robonomics is an informal term used to describe how cyber-physical agents can co-exist within the human economy. One of the projects in that area, called AIRA [10] implemented an overlay p2p network that provided solutions for smart cities and industry. Some of the possible use-cases in robonomics are goods acquisition and delivery services. This business process can be fully automated without the need of any single centralized third party like a bank or delivery service using existing technologies. In the following chapters, theoretical models of such service will be discussed. To conclude this chapter, it is important to mention that the way of building fully autonomous decentralized selling/delivery service includes several questions to resolve, such as: autonomous vehicles organization (ITS system development and regulation problems); smart contract and related ecosystem technology adoption; the problem of decentralized location verification of cyber-physical entity. The last problem is what this paper is about.

2 The Problem

The Global Positioning System (GPS) was developed to help a human with navigation problems. Today, GPS is one of the most important features of a cell phone. It makes possible navigation apps, but not the only. FourSquare entertainment app uses GPS to “award” people for visiting different locations. SnapChat messaging app has the feature that shows where your contacts are currently located. Most probably, these popular applications are not a complete list of GPS use cases. Transportation companies use different GPS enabled proprietary devices that are tracking vehicles for different reasons. GPS is a mature and highly efficient technology, but it has only informational nature. GPS is ideal for personal use to assist yourself in different tasks, but when it comes to using GPS to get another subject's location and verify it for different reasons, then we are not protected from a scam. For example, the above-mentioned transportation companies need to spend money on expensive proprietary devices that track a location using GPS. Why do they need a separate device, while any smart-

phone has a GPS module? Because when the other subject is interested in not providing, at all, or providing a wrong location, it can be achieved relatively easily, unless the device is tamper-proof. This is because the GPS location is just temporary binary data stored in the device memory. For services where a location is economically important, we cannot simply rely on GPS data. For most nowadays services, centralized proprietary location solutions are expensive, but an acceptable choice. But in the development of decentralized autonomous solutions using smart contracts, it will not work, and we will discuss why further.

We will assume the existence of the delivery service based on a smart contract. Some customers want to order package delivery to some location. We will not consider how the contract knows which package the customer wants to get for the sake of problem model simplicity. So, the customer sends the order to the contract with location and time, as well as the payment for the service, which will be deposited and will go to the service provider, as a fee, if the package will be delivered correctly. The contract's role in this scenario is the third party auditor, that needs to hold money on deposit and unlock it for the service provider if the job is done. We assume that the service provider uses autonomous drones to deliver packages. It is necessary somehow to formalize the definition of a completed job in the smart contract. In our case, a completed job means that a specific package is at the specified location at some point of time. Firstly, it seems like the customer can verify the job by himself, by roughly speaking, pressing a button in the app when he took the package from the drone. But such an approach will not satisfy the service provider, because the customer can misuse service by placing orders and not confirming delivery. If the contract cannot check that the job was done, then the deposit will be returned to the customer. So, we can use a GPS module that is attached to the drone to verify that the drone is at the specified location. Then from the contract perspective, the definition of a complete job will be defined as the following algorithm:

- Get the order from the customer with location data and payment
- Get the confirmation from the service provider that the order is accepted
- Wait for the input message with the courier location data from the courier
- Compare the arrived location data with the location data specified in the
- order
- If the data are the same, then unlock the deposit for the service provider.

Hence, right after the service provider confirms the order, it also sends the location specified in the order and gets the customer's money, by not even starting the drone. Maybe, we can use a combination of both parties' confirmation as the definition of a complete job? This will lead to the equivalent of the "only customer" confirmation scheme which means that a customer can still misuse with a service provider. We now understand that 2 parties alone cannot organize themselves without leaving one of the parties vulnerable to "attacks". Hence, without the secure algorithm for verification of

the complete delivery job, we cannot fully automate the location-based services using smart contracts.

Another way to solve the location verification problem in smart contracts is to convert this issue to the problem of trust between client and service and apply blockchain-based reputation models. Such an approach does not provide the framework for autonomous, decentralized robonomic chains, but either advocates service to do everything to make their rating high enough. If the requirement is to build an end-to-end delivery service using decentralized solutions, then such an approach will not work.

3 Related Work

Here we conduct a historical literature review on the topic of the location proof. In one of the first papers about location proof [4] the authors present the algorithm that can securely prove that, assumed to be tamper-proof, device is within the area of a local wireless network. The authors use three roles (location manager, verifier, and prover) as well as the signal round trip time to check if the device is somewhere around. There are several other researches on this topic [5],[6] which mainly focused on the technical aspect of Wi-Fi signals and the methods of gaining the additional information from signals, but these solutions continue to be centralized which will not work in a smart contract architecture. One of the researches [7] proposes a decentralized solution using a graph of hardware-agnostic mobile devices, which can issue “proof of location” to one another, by using short-range communication. The nodes collect signed proofs and combine these into blocks. The solution considers a Proof-of-Stack blockchain to be run by mobile nodes. Considering that this [7] might be the only proposed decentralized solution to the problem of location proof, some of the major drawbacks will be listed here:

- The protocol requires to check if the “prover” is from the “known contact list” of the overlay network. This introduces more complexity and uncertainty. For example, how the peer will know about such a list of the "known contacts" of various location-based services
- The solution requires users run full blockchain nodes which is a hard task for mobile devices in terms of resources
- In the “non-conservative mode”, the peer can verify the distance between two claimed nodes. There are no details of this process. If this procedure is going to be launched on blockchain, then it will be hard for the newcomers to get their proof of location into the blockchain, because most blocks will be mined in most dense areas only. If this is done by using neighbour polling, then this is a resource hard task
- Authors do not provide information about how to use their protocol in smart contracts despite this is the only use case for such a solution.

There are also a few ongoing projects [8], [9] that try to resolve the problem of proof of location in a decentralized way. The Foam project is building an Ethereum overlay network, which will use the specialized hardware to provide location proofs.

4 The Algorithm

In this chapter, we will describe the novel algorithm that solves the problem defined in the second chapter.

4.1 General Flow

Suppose we have a straight street in the urban area. On both sides of the street, there are multistory buildings. There are many private Wi-Fi access points in this area. We will use those as an oracle mesh network to sign the location proofs of different subjects. In Fig. 1, we see a potential topology of Wi-Fi access points where the edges denote that two nodes can communicate with each other directly.

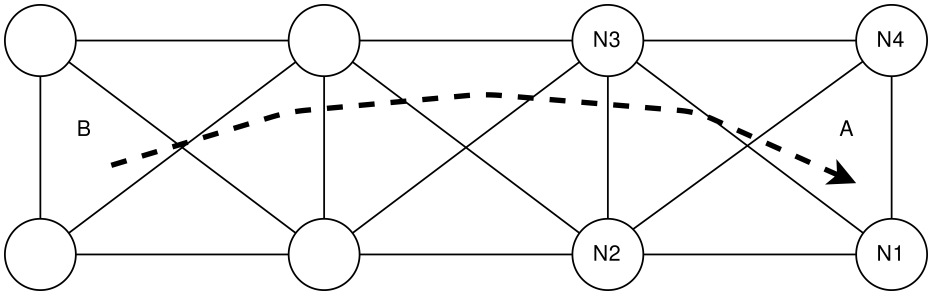

Fig. 1. Topology of Wi-Fi access points.

Point A refers to the customer of the delivery process introduced in chapter two. The point B is a service that will deliver a good. There is also some delivery app (maintained by B) that the customer uses to place the delivery order.

In Fig. 3, we can see the UML sequence diagram of the algorithm. When the service received an order from a new customer *DeliveryRequest(A_pub)* via the delivery app - our algorithm starts. First, the service needs to know if the customer is somewhere within the network boundaries. The network has a public interface with a set of commands. First command is *RequestDKGContract(A_pub)* (short DKGRequest). DKG abbreviation in command's name stands for Distributed Key Generation [11]. It will be described later in the paper. As long as the service is also a network peer, it broadcasts DKGRequest using Wi-Fi probe request. This allows peers to communicate without connecting to Wi-Fi SSID. Probably, Wi-Fi adhoc networks can be used here as well, but to solve the problem defined in this paper, these details can be ignored. The customer in parallel broadcasts his signed beacons to neighbor peers using

Wi-Fi beacon frames. DKGRequest is propagating through the network as long as some peer will receive it together with receiving the customer's beacon frame. DKGRequest also contains the GPS coordinates claimed by the customer. This is needed to protect the service from the malicious customer misusing it. Also, if a node receives a DKGRequest with GPS location that differs from the node's location, it can simply pass it further. All network peers that received DKGRequest together with the customer's beacon start a DKG algorithm. For now, a particular implementation of that algorithm is beyond the scope of this paper. The main goal of DKG is to generate a threshold signature scheme with some *DKG_Pubkey* and a private key for each participant.

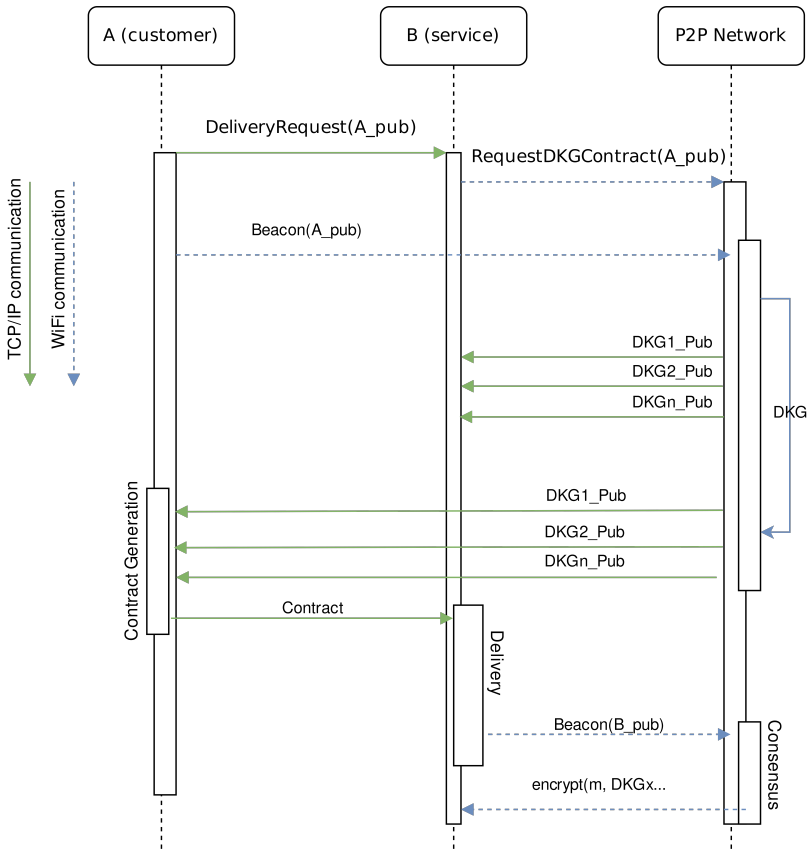

Fig. 3. The algorithm UML sequence diagram.

To collectively sign something using private keys, peers will need to reach some threshold number. For example, ten peers received the customer's beacon. Each of them received his unique private key after the DKG procedure. Now, to sign something, at least six from ten peers need to apply his private key, otherwise, it is "impos-

sible" to produce a valid signature of the current *DKG_Pub* key. The main property of the DKG scheme is that the algorithm requires no third party to help generate keys.

Peers send produced *DKG_Pub* to the customer. The customer generates a smart contract with GPS coordinates and deposits the delivery service fee there. The contract has a simple predicate in it: if *decrypt(m, DKG_Pub) = true*, then release the fee to service. The customer sends this contract to the service. Service checks that the contract is created by some standard interface and if everything is correct, it can start delivery.

When the courier approaches the delivery zone, it starts to broadcast signed beacons. All the nodes that participated in the DKG collect beacons from the courier. When the courier stops, every node encrypts all the collected data with their private keys. The nodes broadcast their encrypted data to each other. Each node needs to verify two facts: first, received signed beacons refers to the original node that called DKGRequest; second, received beacons were not sent from a static position. The first assertion is trivial and essential. The second assertion is an additional verification of the delivery work because otherwise, it is much simpler to mimic the delivery fact by just transmitting the right beacon. By encrypting every node's recorded data, we ensure the uniqueness of every version of the recorded trajectory of the delivery agent. After that, nodes use Lamport's algorithm to achieve consensus (will be described in the next section).

After the consensus is achieved, nodes sign the verdict with their DKG private keys and send it to the delivery service. If the consensus is not achieved, then the message will not be signed. Considering that, the threshold also will not be reached, then there is no chance to get *decrypt(m, DKG_Pub) = true*. Otherwise, if the threshold is reached, the delivery service could withdraw the fee from the contract with the signed message from the network.

4.2 Consensus Algorithm

We can use two approaches to achieve consensus on whether the delivery was done correctly. The first is to achieve consensus only between nodes N1, N2, N3, and N4 (Fig. 1). This means DKGs will be triggered only between nodes that are located directly on the target location. The second is to try to achieve consensus in all zones where there was a delivery path. This is a topic for separate research, but in the current paper, we only describe the first approach. We consider our oracle network to be Byzantine fault-tolerant (BFT) and can handle no more malicious/faulty nodes than in Eq. (1), where m is the total number of nodes [1].

$$\lfloor (m-1)/3 \rfloor \tag{1}$$

Eq. (1) states that, if nodes N1, N2, N3, and N4 want to reach consensus on the fact of delivery, then at least 3 nodes must be loyal. The fact that each node can be malicious, it can try to break protocol in any possible direction, potentially stopping the network to reach consensus. This implies that the only way to avoid this is to implement BFT consensus. For now, we can assume that the network will use the BFT consensus algorithm from Lamport's paper [1].

4.3 Alternative Scenarios and Security

To ensure that the algorithm will work in a hostile environment we need to make sure every node has at least 2/3 loyal neighbors. Let's call it "condition 1". This condition will ensure the possibility of final consensus according to Lamport's algorithm. There is one problem which is called "Sybil attack". Theoretically, nothing prevents one malicious node from acting as many nodes. This will break any consensus because condition 1 will not be accomplished. In order to distinguish nodes, we can utilize information from Wi-Fi CSI (Channel State Information). This allows a node to calculate signal location in space, which implies the identification of nodes based on public key and CSI.

If the customer is malicious, it is relatively easy for him to fail condition 1 using a set of real physical nodes in some area. For example, DKG occurred in such an area, where more than 1/3 nodes are owned by a malicious customer. When service delivers a package in this area, malicious nodes simply will not sign the message for the contract. This allows a malicious customer to misuse with a customer and waste his time/money.

We still can make a dishonest customer's life more uncomfortable. We need to introduce the risk that attackers can lose deposited service fees. If that risk will be high enough, it will respectively reduce the chance of someone misusing the service. There is no way for the service to distinguish malicious contracts/nodes before the consensus phase. This means that we should search for a probabilistic solution. For example, we can let service deliver on adjacent sectors to a customer area. Let's imagine that the malicious area has more than 1/3 dishonest nodes, but probably there are still at least a few loyal nodes. Now, each of them will set up DKG. There is a high probability that at least one DKG area will be loyal. Ultimate contract might have the following assertion: *decrypt(m, DKG1_Pub) = true or decrypt(m, DKG2_Pub) = true or...* This means for service that it can choose in which area to deliver and if in some area consensus will not be reached it can still try a few adjacent

The last problem we need to solve is how the service will know, which *DKGn_Pub* must be in the ultimate contract that he receives from the customer? All loyal nodes which received DKGRequest at the beginning will essentially send their *DKGx_Pub* to service, so that service knows there are multiple areas that are ready to do loyal consensus. Now, if the customer will not include some of the DKGs into the contract,

the service simply might ignore that contract and decline delivery. This means protocol protects, to some degree, service from malicious customers. What about malicious services? The only attack which malicious service can do is to fully blockade the customer's area with malicious nodes. Probably this attack is too expensive to organize compared to what malicious service could "earn" by stealing customers' service fees.

4.4 Future Research

There are several directions for further research on this topic. First of all, the analysis of CSI-based solutions that can calculate signal source location in space. In the algorithm described in the current paper, those solutions are vital for node identification and delivery verification. Secondly, analysis of existing synchronous and asynchronous DKG protocols implementations. And of course analysis of existing BFT consensus algorithms.

5 Conclusions

A novel theoretical algorithm that solves proof of the location problem in smart contracts was introduced. This algorithm assumes the decentralized p2p network of Wi-Fi nodes that forms a mesh network. The network by itself does not store any history of the location proof and works only on demand. The use of the distributed key generation algorithm allowed to establish a smart contract between two parties. The network monitors subject beacons and uses byzantine fault-tolerant algorithm to achieve consensus on the location proof of the subject.

References

1. Lamport, L., Shostak, R., & Pease, M. (1982). The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems, 4(3), 382–401. doi:10.1145/357172.357176
2. Nakamoto, S. Bitcoin: a peer-to-peer electronic cash system. 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf
3. V. Buterin, "A next-generation smart contract and decentralized application platform". 2014.
4. B. Waters, E. Felten, "Secure, private proofs of location". 2003. [Online]. Available: https://www.cs.princeton.edu/research/techreps/TR-667-03
5. Javali, C. et al., 2016. I Am Alice, I Was in Wonderland: Secure Location Proof Generation and Verification Protocol. 2016 IEEE 41st Conference on Local Computer Networks (LCN). Available at: http://dx.doi.org/10.1109/lcn.2016.126.
6. Luo, W. & Hengartner, U., 2010. VeriPlace. Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '10. Available at: http://dx.doi.org/10.1145/1869790.1869797.

7. Amoretti, M. et al., 2018. Blockchain-Based Proof of Location. 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C). Available at: http://dx.doi.org/10.1109/qrs-c.2018.00038.
8. Foamspace Corp, “FOAM Whitepaper”. 2018. [Online]. Available: https://foam.space/publicAssets/FOAM_Whitepaper.pdf
9. A. Trouw, M. Levin, S. Scheper, “The XY Oracle Network: The Proof-of-Origin Based Cryptographic Location Network”. 2018. [Online]. Available: https://docs.xyo.network/XYO-White-Paper.pdf
10. S. Lonshakov, A. Krupenkin, A. Kapitonov, E. Radchenko, A. Khassanov, A. Starostin, "Robonomics: platform for integration of cyber physical systems into human economy". 2018.
11. Torben Pedersen. A threshold cryptosystem without a trusted party. In Eurocrypt ’91, pages 522–526, 1991. LNCS No. 547.