Decentralised location proof system

Conor Taylor

B.A.(Mod.) Computer Science
Final Year Project, April 2016
Supervisor: Stephen Barrett

Contents

1 Introduction **1**
- 1.1 Project and motivation . . . 1
- 1.2 Report outline . . . 2
  - 1.2.1 Background . . . 2
  - 1.2.2 Design . . . 2
  - 1.2.3 Evaluation . . . 2
  - 1.2.4 Conclusion and Future Work . . . 2

2 Background **3**
- 2.1 Proving location . . . 3
- 2.2 Distributed and decentralised systems . . . 4
- 2.3 Existing location proof systems . . . 6
- 2.4 Blockchain . . . 9
- 2.5 Design goals . . . 10
  - 2.5.1 Central services . . . 11
  - 2.5.2 OTIT . . . 11
  - 2.5.3 Threats . . . 12

3 Design **14**
- 3.1 Identities . . . 16
  - 3.1.1 Nonce list . . . 16
  - 3.1.2 Identity duplication . . . 17
- 3.2 Transactions . . . 18
  - 3.2.1 Key Lists . . . 19
  - 3.2.2 Transaction creation . . . 19
  - 3.2.3 Publishing to the blockchain . . . 20
  - 3.2.4 Key Packets . . . 21
  - 3.2.5 Aborting exchanges . . . 21

3.3 Verification 23

4 Evaluation **26**
4.1 OTIT conformance 26
4.1.1 Chronological 26
4.1.2 Order preserving 27
4.1.3 Verifiable 27
4.1.4 Tamper evident 27
4.1.5 Privacy preserved 28
4.1.6 Selective in-sequence privacy 28
4.1.7 Privacy protected chronology 29
4.1.8 Convenience and derivability 29
4.2 Threat analysis 29
4.2.1 False presence 29
4.2.2 Malicious intruders 30
4.2.3 Curious users 30
4.2.4 Malicious applications 31
4.2.5 Eavesdroppers 31
4.2.6 Wormhole attacks 31
4.2.7 Weak identities 31
4.2.8 False timestamping 32
4.2.9 Implication 32
4.2.10 False assertion 32
4.2.11 Proof switching 33
4.2.12 Relay attack 33
4.2.13 Sybil attack 33

5 Conclusion and Future Work **35**
5.1 Future work 35
5.1.1 Verification techniques 35
5.1.2 Privacy and verifiability 35
5.1.3 Sybil attack 35
5.2 Conclusion 36

List of Figures

2.1 Distributed location proof system 5
2.2 Decentralised location proof system 5
2.3 Hardware-based proof system (Adapted from Brassil et al. [1]) 6
2.4 Distributed proof system on access points (Adapted from Luo et al. [2]) 7
2.5 Proof request with group signature (Adapted from Luo et al. [2]) 8
2.6 Proof system using witnesses (Adapted from Khan et al. [3]) . 9
2.7 Blockchain overview 10

3.1 Design and operation overview 15
3.2 Identity generation from Nonce List 17
3.3 Honest, successful exchange 18
3.4 A proof chain with transaction T_{m-1} hidden 21
3.5 An aborted transaction due to malicious node M 22
3.6 Verification request 23
3.7 Alibi chain relationship tree 24

4.1 A proof chain with uncredible alibis 30

Chapter 1

Introduction

1.1 Project and motivation

Location verification is the process of verifying whether a *node* (computer) is physically present at a location it claims to be. There are two general categories of location verification; *observation* and *proof*. In location observation, a user's location is determined by observing certain properties of the user, its IP address for example. This is a cheap but very unreliable method of location verification. In location proof, a user's location can be proven reliably.

Existing location proof systems attempt to use centralised, trusted "authoritative" nodes to provide proof of another node's location. These approaches require investment in infrastructure, and are subject to privacy violation and denial of service attacks. This project aims to present a *decentralised* solution to this problem. A decentralised location proof system is a system in which there is no "authoritative source" trusted and relied upon to provide and store sensitive location information.

This project describes a decentralised location proof system that is capable of operating on *mobile nodes* (mobile devices). I propose a design in which location proofs are obtained using other untrusted mobile nodes as *alibis*. Proofs will be created as two mobile nodes communicate over an ad-hoc bluetooth network, transfer encrypted location information, and publish it on a public append-only bulletin board, known as a *blockchain*. The decentralised nature of the system means that there is no single point of failure, no entity controlling the security of every node's location proofs, and no entity

capable of violating another node's privacy. This is because in a decentralised system, no node is "in charge", and no node has more authority in the system than any other node.

1.2 Report outline

This report consists of four additional chapters, listed briefly below.

1.2.1 Background

Chapter 2 begins with an introduction to the problem space. A brief explanation of decentralised and distributed networks, as well as the blockchain is then presented, in the context of location proof systems. The chapter closes with an explanation of the design goals defined for this project.

1.2.2 Design

Chapter 3 explains the design of the decentralised location proof system in detail. It begins with an overview of the entire system, and then details the design of each specific part.

1.2.3 Evaluation

Chapter 4 presents an evaluation of the system described in the design, with respect to the goals laid out in Chapter 3.

1.2.4 Conclusion and Future Work

Chapter 5 presents the conclusions of the project, and details interesting areas for possible future work relating to the project.

Chapter 2

Background

2.1 Proving location

As an increasing amount of personal information is accessible on the internet and therefore on mobile devices, the security that previously existed by requiring physical interaction between humans to transfer sensitive data is lost.

Location proof is a large part of that security. There is currently no ubiquitous method of *proving* a mobile user's location at a given time over the internet. Many companies, such as Netflix, use GeoIP databases such as MaxMind [6] to estimate a user's location based on its IP address. This is a location observation technique rather than a proof. It is therefore inaccurate and easily bypassable using proxy servers or VPNs.

One newer, more advanced technique of location observation is Constraint-Based Geolocation [7]. This is an active geolocation technique, compared with GeoIP verification which is static. It estimates the location of a users IP address by calculating the latencies between the user's machine and multiple surrounding machines with fixed, known locations. However, even advanced systems like these can easily be bypassed, using IP spoofing techniques, proxy servers, VPNs, or by using anonymising networks such as TOR [9].

Better solutions to this problem exist. *Location proof systems* allow users of the system to *prove* their location to another user, by creating *location proofs*. The structure of a location proof varies from system to system. In essence, a location proof is a piece of data, potentially signed or encrypted by another party, which attests the location of a user at a specific time.

Examples of location proof systems are discussed in detail in section 2.3.

2.2 Distributed and decentralised systems

A *distributed* system is a system which distributes the computation of a task across multiple nodes (computers) connected over a network. These nodes are assigned a task from a central node, and then work together and communicate to complete the task by passing messages to each other over the network [10].

For example, Google search indexes are not computed on just one node, rather on a network of nodes that communicate by sending messages to each other. Google therefore *distributes* the task of calculating search indexes across multiple nodes. This has two main advantages; It is less time consuming, because different index calculations can be computed on different nodes at the same time, and it is scalable, because nodes can be added or removed from the network dynamically, to meet demand.

A *decentralised* system is similar to a distributed system, except no node is in charge. Nodes in a decentralised network still communicate by passing messages to each other over the network, but unlike a distributed system, they are not assigned tasks by a central or master node. Bitcoin [12] is a good example of a large-scale decentralised system. Two nodes in the Bitcoin network can choose to send money between themselves, creating a transaction. They make the details of that transaction public. The two nodes therefore create and publish the transaction without coordination from a central node.

In the context of this project, a distributed location proof system is considered to be a proof system where the task of facilitating user proof request and creation is distributed across multiple *worker nodes*, which are controlled and coordinated by a central node. This allows a higher volume of proofs to be created concurrently compared with using a single node, while maintaining centralised and simple control over the system. This is shown in figure 2.1.

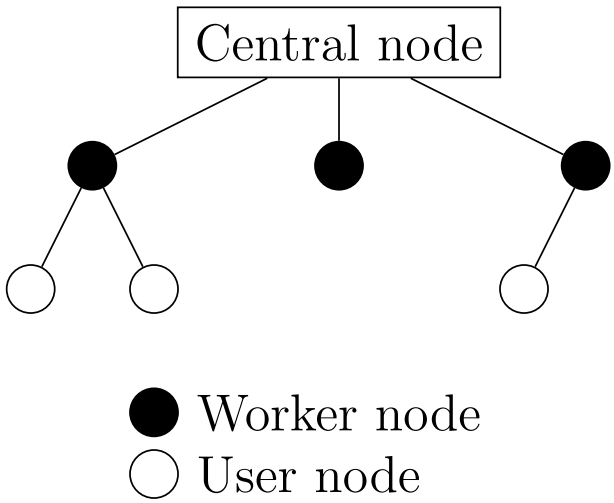


Figure 2.1: Distributed location proof system

A decentralised location proof system is considered to be one where no node is in charge of controlling or coordinating the system. Nodes can create and publish location proofs between themselves, without the need for a central third party to coordinate their interaction (figure 2.2). This makes coordination more complex, but means that no node is considered to be "in charge" of the system.

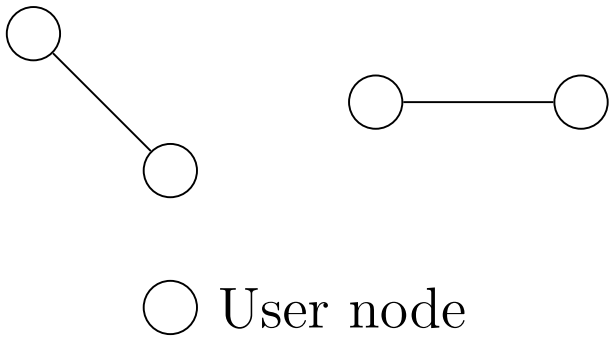


Figure 2.2: Decentralised location proof system

2.3 Existing location proof systems

Location proof systems are expected to be accurate and tamper-proof. For this reason, existing solutions [1, 2, 3] have chosen to use a central authority to issue proofs, or to regulate proof issuance.

A hardware technique [1], developed by Brassil et al. of HP Laboratories, operates by supplementing existing Wi-Fi access points (*AP's*) with *femtocells*. A femtocell [13] is a small cellular antenna that connects to a mobile network operator (such as Vodafone) via the Internet, which allows mobile networks (e.g. networks used for calls and texts) to be extended on demand. Location verification over the internet is made possible by determining which femtocell a mobile node is connected to as it transfers data over Wi-Fi. This is possible because the central server in the location verification system has access to the mobile operator's user data.

This solution requires investment in additional hardware to supplement existing Wi-Fi access points, and requires access to mobile providers' user databases to identify users' locations (see figure 2.3). Note that the two lines between the user and the AP in figure 2.3 indicates that the user is connected to the AP *and* its attached femtocell.

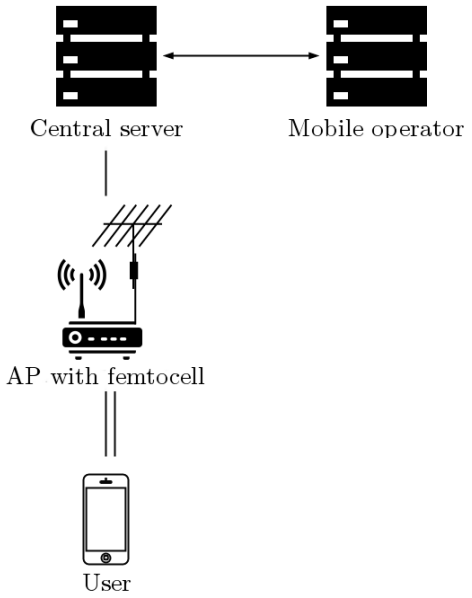


Figure 2.3: Hardware-based proof system (Adapted from Brassil et al. [1])

The use of a centralised system, as described in figure 2.3, creates security, privacy and vulnerability issues. An attacker who succeeds in compromising the security of the central server can create unlimited false location proofs for itself, or violate the privacy of the users of the system, and potentially track their location. This is because in this system, location proof data resides with the central server. This means that the user has no control over the security of their proofs, and an attacker could seize them. The central system architecture is also vulnerable, in the sense that a resource availability attack such as a DDoS attack [14] could render the central architecture unavailable, making location verification unavailable.

Luo et al. propose a system that uses software installed on Wi-Fi access points to allow users to create location proofs [2]. In their system, each access point is given a *group signature* by a central server, and can sign location proofs for users (figure 2.4).

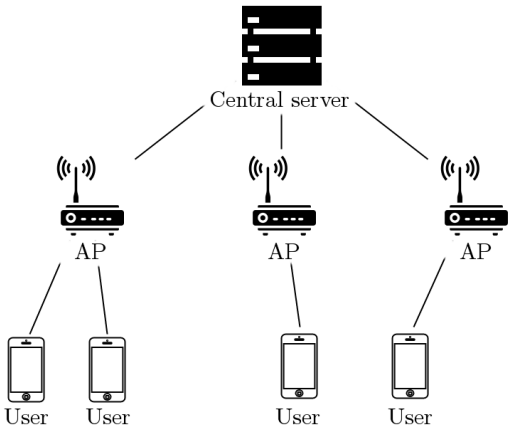


Figure 2.4: Distributed proof system on access points (Adapted from Luo et al. [2])

Users can request a location proof from any access point, and receive a proof encrypted by the AP with the group signature, as shown in figure 2.5. This can then be submitted to a Verifier.

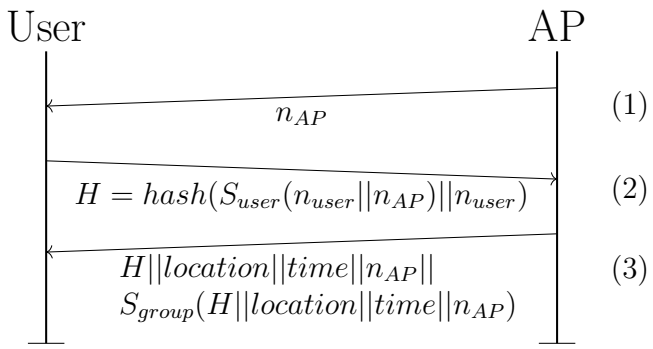


Figure 2.5: Proof request with group signature (Adapted from Luo et al. [2])

The operation of the system shown in figure 2.5 is described below.

(1) The access point generates a nonce n_{AP} and sends it to the user.

(2) The user generates its own nonce n_{user}, and responds with a one-way hash of its nonce and the AP's nonce, encrypted with its own secret key.

(3) The AP generates a location proof, encrypted with S_{group}, the system group signature. This proof now resides with the user, and can be used to prove its location.

This system creates *proactive* location proofs. A proactive location proof is one which is created before it is needed. The user creates application-independent location proofs, and can use them at a later time with any application(s) it chooses.

A system proposed by Khan et al. takes a different approach, by using other users as witnesses to a location claim [3]. In this system, a *witness* physically located near the user is used by the central server to verify the user's location claim (figure 2.6). Like the model proposed by Luo et al., this system allows the user to have control over their own privacy, as the user owns the location proof.

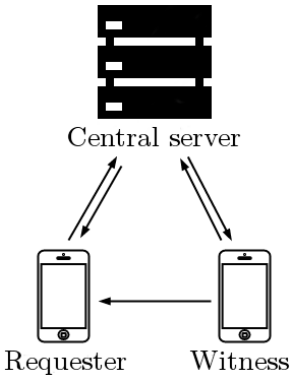


Figure 2.6: Proof system using witnesses (Adapted from Khan et al. [3])

2.4 Blockchain

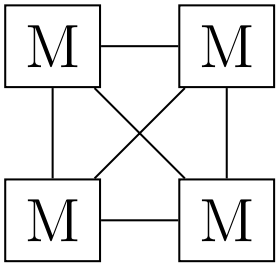
In Bitcoin [12], the entire transaction history is stored in the *blockchain*. This is a decentralised, append-only ledger (database) [11]. The blockchain is maintained by a network of *miner* nodes competing to collect transactions into a new block, sign that block by solving a difficult puzzle known as a *proof-of-work*, and append it to the ledger. It is used in Bitcoin to enforce consensus; every miner in the network agrees that all blocks in the blockchain are valid, due to the presence of their associated proof-of-work.

A proof-of-work is a solution (a numerical value) to a problem which is computationally expensive to solve, but computationally cheap to verify that the solution is correct. As blocks are added to the blockchain, the amount of work required to generate the proofs-of-work for the entire chain increases. Because proof-of-work solutions also incorporate some information about the previous block in the chain, changing data inside a block in the blockchain would not just require regeneration of the proof-of-work for that block, but for all successive blocks as well. A malicious node attempting to change the transaction history would have to solve proofs-of-work at a greater rate than the rest of the network combined in order to succeed. Therefore the blockchain is tamper-proof, assuming that the majority of the CPU power in the miner network is controlled by honest miner nodes [12].

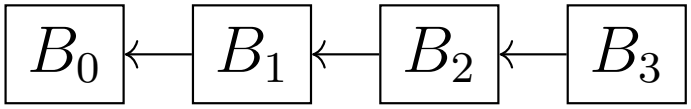
To add a transaction to the blockchain, a user will send the transaction to a miner node. That miner node propagates the transaction to other miner

nodes in the network, and they each try to include it in the next block by solving the proof-of-work. A simple overview of the components in the system is shown in figure 2.7.

Once a proof-of-work has been solved, the solution is distributed to all other miner nodes in the network to prove that the solver expended significant computational effort to solve it. This is a proof of resource ownership, meaning that by solving the proof-of-work, the miner node has shown that it has significant computational resources and is therefore considered a valid miner node. Once other miner nodes in the network verify that the proof-of-work is correct, the block (and therefore all transactions contained in it) is considered the newest block in the tamper-proof chain, and miner nodes begin working on the next block.



(a) Miner node network



(b) Blockchain

Figure 2.7: Blockchain overview

2.5 Design goals

Before designing the system, a number of goals must be defined and an attempt made to satisfy them in the design. The design goals of this project are listed below.

1. The system must be *privacy preserving*. It must not be possible for

a malicious user to obtain any additional information from an honest user other than the information the honest user intended to release.

2. False location proofs must be detectable. It must be possible to detect any location proofs attempting to prove false information.

3. The system must be decentralised, and cannot rely on any central nodes. This is discussed in section 2.5.1.

4. The design must conform to the desirable properties defined in OTIT (discussed in section 2.5.2).

5. The system must not be vulnerable to any known threats (section 2.5.3).

2.5.1 Central services

Location proof systems which rely on central nodes, such as those discussed in section 2.3, have a fundamental weakness. These systems assume that the central node is honest, always available, and cannot be compromised by an attacker. Since the central node is in charge of coordinating the system, compromising it allows an attacker potentially unlimited control over the system. This completely violates a user's privacy, as the central node may contain sensitive information about users and their location, or a means of retrieving sensitive information, such as decryption keys. Users of the system also have to assume that the central node is available at all times, otherwise the system is considered unavailable, or "down".

These problems do not occur in a decentralised location proof system, because no node is considered to be "in charge". If no node is in charge, any node(s) can join or leave the network without altering the state of the network, or making any services unavailable. This lack of authority prevents privacy breaches, but presents other issues. System coordination, which is easily maintained by the central node in a centralised system, is now maintained by each participant in the decentralised system, who may or may not all be honest.

2.5.2 OTIT

Khan et al. present the *OTIT* model, a "model for designing secure location provenance" [4]. It defines the following 8 desirable properties of a location

proof system:

Chronological: Location proofs should be ordered according to the sequence of their creation.

Order preserving: The order in which the proofs are entered into the proof chain should always be maintained.

Verifiable: A Verifier should be able to verify or reject the claim by the user that it has visited the given location(s).

Tamper evident: A Verifier should be able to detect if the proof chain has been tampered with or modified by an unauthorised party.

Privacy preserved: When the proof chain is verified for specific location proofs, privacy is preserved for the other location proofs within the chain.

Selective In-sequence privacy: The user should be able to choose not to reveal some location proofs within the proof chain when seeking verification.

Privacy protected chronology: The system should ensure that the user does not hide important information from the items within the subset of the proof chain.

Convenience and derivability: The system should ensure that the user does not burden the Verifier with a large amount of data to analyse in order to verify its location.

2.5.3 Threats

A number of related publications [2, 3] have identified various potential threats (or attacks) on location verification systems. These are outlined below.

False presence: A dishonest user attempts to obtain a location proof for a false location.

Malicious intruders: A malicious intruder offers to help another user prove a false location, but does not prove a false location for itself.

Curious users: A curious user learns another user's identity by monitoring its location proofs.

Malicious applications: The application that the user is proving its location to takes advantage of the user's information.

Eavesdroppers: An eavesdropper intercepts or modifies communication between two honest users.

Wormhole attacks: An attacker records network traffic in one location and replays it in another location (at another time).

Weak identities: A user gives its public key or identity to another colluding user in order to create a location proof.

False timestamping (backdating, future dating): In a backdating attack, the attacker creates a location proof with a past timestamp. In future dating, the attacker creates a location proof with a future timestamp.

Implication: A malicious user, or group of colluding malicious users, creates a false location proof for an honest user.

False assertion: Two malicious users collude to create a false location proof for themselves.

Proof switching: A malicious user modifies one of its existing honest proofs to spoof a false location.

Relay attack: A malicious user creates a location proof with an honest user by relaying its requests through a colluding *proxy user*, who is physically present at a location that the malicious user wishes to create false proofs at.

Sybil attack: A Sybil attack occurs when a single malicious user of the system generates multiple *pseudoidentities* and masquerades as multiple users [5].

Chapter 3

Design

The location proof system proposed in this project allows *mobile nodes* to create location proofs by initiating an *exchange* with another anonymous mobile node. Mobile nodes, physically moving around, advertise their identity over a short range, ad-hoc network such as bluetooth. When two mobile nodes are in close physical proximity, they initiate an exchange in the background over the ad-hoc network. On completion of the exchange, both nodes can generate their own *transaction*, a cryptographically secure proof of location, attested by the other mobile node (the *alibi*), and make it public.

To create a transaction, two mobile nodes anonymously exchange their GPS coordinates and current timestamp over the ad-hoc network. Both nodes then check that these parameters do not differ from their own observed parameters by more than some value ϵ. Once satisfied, they each create an encrypted, privacy-protecting transaction logging their location and the identity of the the alibi used to create the transaction. This transaction is published onto a public, append-only bulletin board known as a *blockchain* [11]. Once given permission from a mobile node, a *Verifier node* can consult the data in blockchain and determine whether or not the mobile node is present at its claimed location.

Any mobile node acting as an alibi in a transaction will share details of its most recent transactions with the other mobile node in the exchange. This means that when a mobile node seeks verification from a Verifier node, the Verifier can examine the mobile node's alibis, and each of the alibis' recent alibis, forming a large tree of connections which can be used to verify or reject the mobile nodes claimed location (see figure 3.7). Only the owner of the transaction can give permission to a Verifier to check its location history,

by providing the Verifier with the keys needed to decrypt the transactions.

Figure 3.1 provides an overview of the operation of the proposed system.

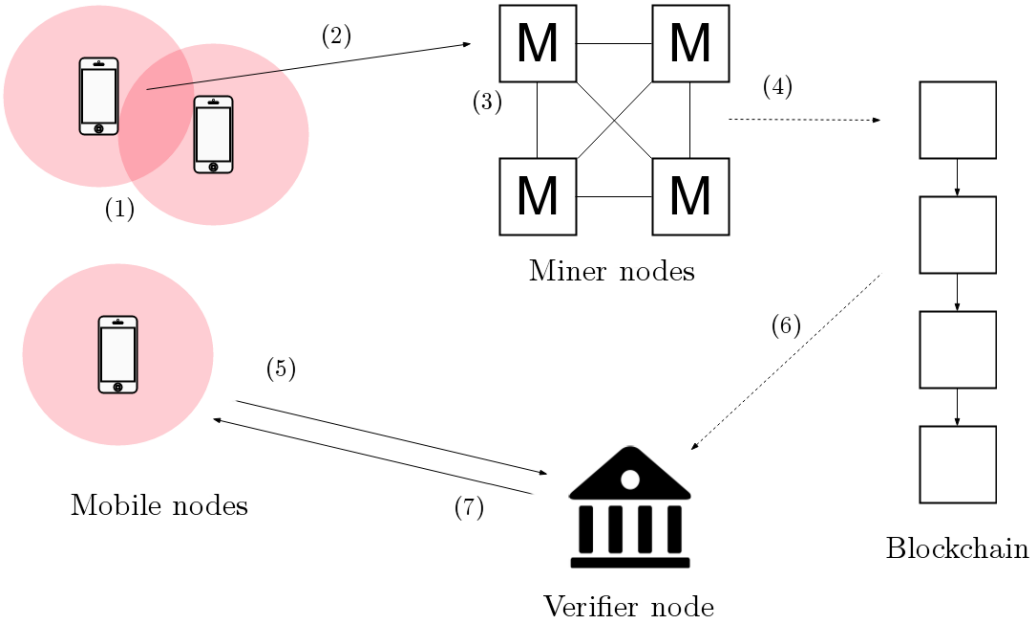


Figure 3.1: Design and operation overview

1. Two mobile nodes come in close physical proximity and create an ad-hoc network, over which they initiate an exchange and each create a transaction.

2. Both mobile nodes publish their own transaction to a Miner node.

3. The Miner node who received the transaction distributes it to the other miner nodes in the network.

4. One Miner node solves the proof-of-work for a block of transactions, and the new block is appended to the blockchain.

5. A mobile node requests verification from a Verifier node (request structure is explained in section 3.3).

6. The Verifier node refers to the public blockchain to examine the requesting node's transactions and determine their validity.

7. The Verifier accepts or rejects the mobile node's claimed location based on its analysis of the transactions in the blockchain.

3.1 Identities

In order to preserve users' privacy, a user will only ever use an identity for one transaction, before generating a new one. This prevents curious users from watching the public blockchain for a known identity and tracking it. However, to prevent identity theft, it is important for a Verifier to be able to prove that a mobile node is the owner of each identity. This is achieved using a public/private key pair for each mobile node.

When a mobile node first joins the network, it generates a public/private key pair. To maintain anonymity, the mobile node will use the public key to encrypt *nonces* to create *identities*, and use these to identify itself in a transaction. A different identity will be created for each transaction. During the verification stage, a node will provide a Verifier with its public key, along with the list of the nonces used to generate its n most recent transactions (see figure 3.6). This allows a Verifier to calculate all of the node's identities, and prove that they were all created by the same node. A Verifier can then retrieve the transactions associated with those identities from the blockchain. Verifiers will also ensure that a node owns the private key associated with its provided public key, to prevent a weak identity attack.

3.1.1 Nonce list

A *Nonce List* is simply a list of all of the nonces a mobile node has used so far to generate identities. When a mobile node wishes to generate a new identity, it must first generate a new random nonce which is then encrypted to create the identity. The nonce is appended to the mobile node's nonce list, so it can be used later during verification (see section 3.3).

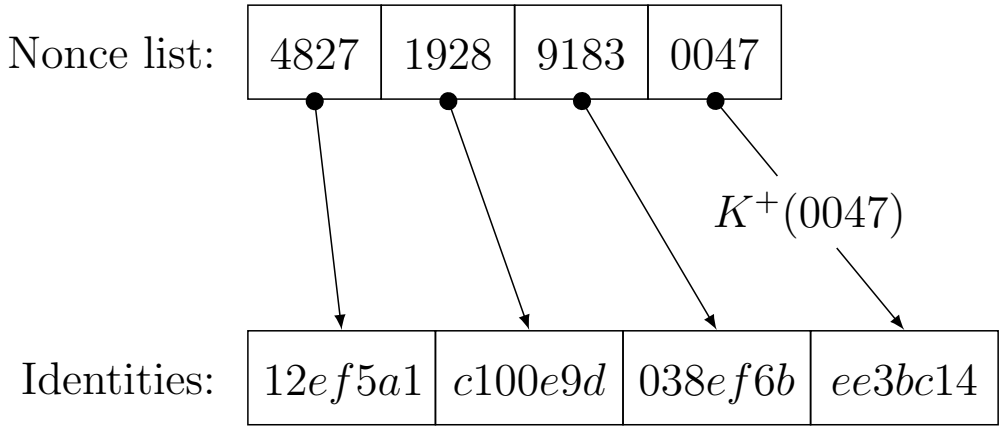


Figure 3.2: Identity generation from Nonce List

In figure 3.2 above, nonce 0047 is encrypted with K^+, a node's public key, to create identity $ee3bc14$. By providing the nonce list and K^+, a node can prove that it created all of the identities in the identities list. It must also verify that it owns K^-, its private key.

3.1.2 Identity duplication

Since all identities are generated from random nonces, there is nothing to prevent two nodes from generating the same identity and using it in different transactions. It is therefore possible that during verification, a Verifier node may find multiple transactions with the same ID while searching the blockchain. Verifiers are provided with decryption key from the requesting node for the transactions they are searching for, which will only successfully decrypt one of the transactions with the duplicate ID. A Verifier will try to decrypt each one until it finds the one that can be decrypted successfully.

3.2 Transactions

During an exchange between two nodes, a number of different items must be shared and calculated. Figure 3.3 describes and explains an honest, successful exchange, assuming an ad-hoc network has already been set up between nodes A and B.

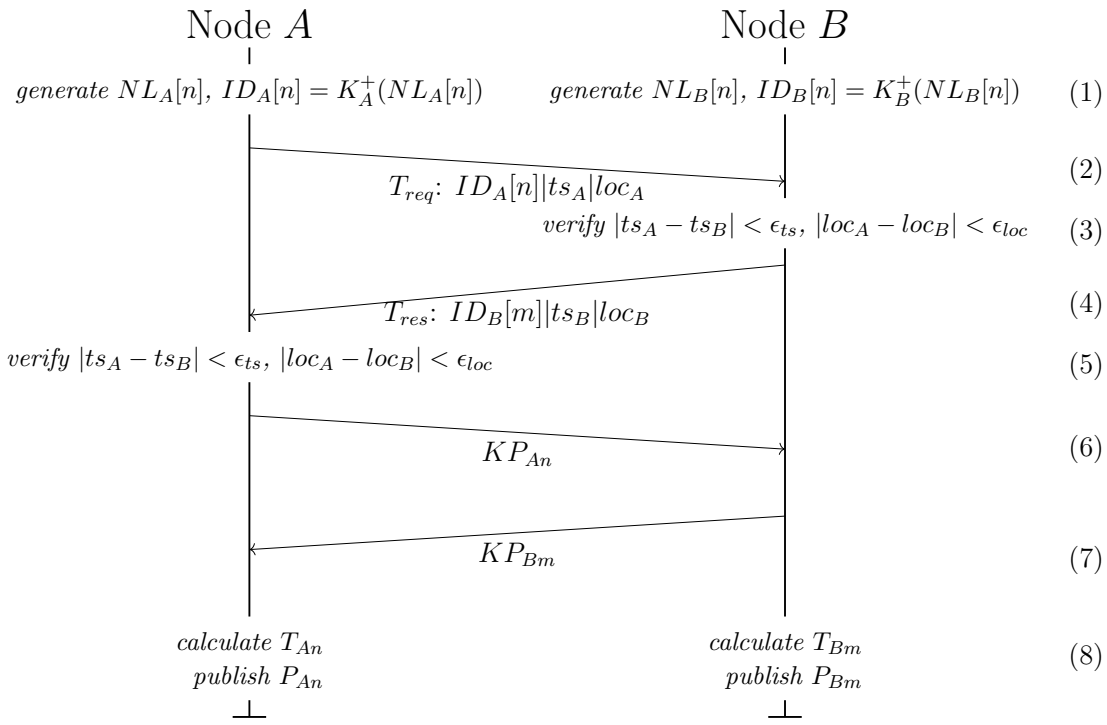


Figure 3.3: Honest, successful exchange

(1) Before the exchange begins, both nodes generate a new entry in their nonce list (e.g. $NL_A[n]$), and generate a new identity from that nonce (e.g. $ID_A[n]$).

(2) Node A sends its identity to node B, along with its observed current timestamp ts_A and current GPS location loc_A.

(3) Node B verifies that the observed timestamp and GPS location received from node A are within an acceptable difference from its own observations. This difference is defined by some system constant ϵ.

(4) If node B verifies the parameters received from node A, it will respond with its own identity and observations for node A to verify.

(5) Node A verifies that the parameters received from node B differ by no more than ϵ from its own observed parameters.

(6) Node A sends KP_{An}, its *nth Key Packet*. Key Packets are used to provide alibi credibility, and are discussed in section 3.2.4.

(7) Node B responds with its own Key Packet.

(8) Both nodes generate their own transaction T, using their alibi's Key Packets. These are then used to generate P, the data published onto the public blockchain.

3.2.1 Key Lists

When a node is initialised, it must begin to record two *Key Lists* for itself. A *Key List* is a list of encryption keys previously used for transactions. For example, $KL_{AT}[n]$ and $KL_{AL}[n]$ are the two keys used to encrypt node A's nth transaction and matching chronological link, respectively. These *Key Lists* are assumed to have been initialised and populated before the transaction described in Figure 3.3 begins.

3.2.2 Transaction creation

Once nodes have exchanged *Key Packets*, their communication is finished and they can close their ad-hoc channel. Each node creates a new transaction, T, and generates data P to publish onto the public blockchain. The following transaction will be created by node A, after communicating with node B:

$$T_{An} = KL_{AT}[n](ts|loc|ID_A[n]||ID_B[m]|KP_{Bm})$$

Where:
n is node A's current transaction number (i.e. node A has published $n-1$ transactions before now).
$\mathbf{KL_{AT}}$ is node A's transaction *Key List*, used to encrypt transactions. $KL_{AT}[n]$ is the *nth* entry in the Key List.
ts is the timestamp of the transaction, as recorded by node A.

loc is the GPS coordinates of the transaction, as recorded by node A.
$\mathbf{ID_A[n]}$ is A's *nth* identity, used for this transaction only.
$\mathbf{ID_B[m]}$ is B's *mth* identity.
$\mathbf{KP_{Bm}}$ is B's *Key Packet* at time m. Its exact contents are discussed in more detail in section 3.2.4

3.2.3 Publishing to the blockchain

After creating T_{An}, node A must publish it to the public blockchain. The data published must be searchable in order to provide a Verifier with some way of identifying specific transactions. Nodes use their current ID as the identifier for a transaction.

Backwards-chaining is an important part of verifiability. A Verifier must be able to prove that the proof chain provided by a mobile node has not been chronologically re-ordered. This was designed to satisfy the "order preserving" property of OTIT [4]. Each transaction must therefore be accompanied by a link to the node's chronologically previous transaction. This link is encrypted with $KL_{AL}[n]$, node A's *nth link* key list, to prevent a curious user from monitoring the public blockchain to observe a user's transaction publishing activity. The timestamp ts_A is also included in the backwards-chaining data, required to verify the "order preserving" property of OTIT. Node A's transaction data P_{An} will therefore be in the form:

$$P_{An} = ID_{An}|KL_{AL}[n](ID_A[n-1]|ts_A)|T_{An}$$

To publish P_{An}, node A must send P_{An} to a miner node, who will add it to its *pending transaction* queue, then send it to other miners in the network to add it to their queues. The transaction will be signed into the next block in the blockchain, once the next proof-of-work is solved.

Node B, A's alibi for the above transaction example, will publish P_{Bm} after the exchange. If node B's transaction is not available on the blockchain, node A's transaction will be considered invalid and disregarded during verification. A transaction T_A in the blockchain must be accompanied by a reference to some valid alibi transaction T_B, and T_B must reference T_A, otherwise the proof is not considered credible by a Verifier.

3.2.4 Key Packets

A key packet is more than simply a list of keys that a mobile node has used to encrypt its transactions; in order to preserve the "selective in-sequence privacy" property of OTIT [4], a user must be able to choose transactions that it does not want others to be able to decrypt. However, in order to simultaneously preserve OTIT's "privacy protected chronology" property, the transaction backwards-chaining must not be broken. This is why two separate *Key Lists* are maintained; KL_{AL} is used to encrypt the backwards-chaining link and transaction timestamp, while KL_{AT} encrypts the transaction data. For example, if node B does not want to reveal transaction T_{Bm-1}, it would use the following key packet:

$$KP_{Bm} = (\{KL_{BL}[m], KL_{BT}[m]\}, KL_{BL}[m-1], \{KL_{BL}[m-2], KL_{BT}[m-2]\})$$

In this case, $KL_{BT}[m-1]$ has not been included with $KL_{BL}[m-1]$, in order to protect the privacy of this transaction. This means that a node who receives this key packet will not be able to decrypt B's transaction T_{Bm-1}, but can still prove that a transaction exists at that time and in the correct sequence, as B will still provide $KL_{BL}[m-1]$ (see figure 3.4). This may be needed, for example, if a user does not want to allow anyone to decrypt any transactions it has created within 5km of its home, in order to protect its privacy.

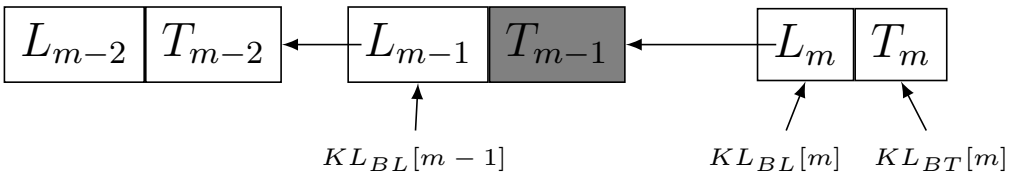


Figure 3.4: A proof chain with transaction T_{m-1} hidden

3.2.5 Aborting exchanges

A node may decide to abort an exchange if it thinks it is in contact with a malicious node. In figure 3.5, node A encounters a malicious node M, who tries to spoof its location to node A. Node A will abort the exchange once it notices that $|loc_A - loc_M| \geq \epsilon_{loc}$.

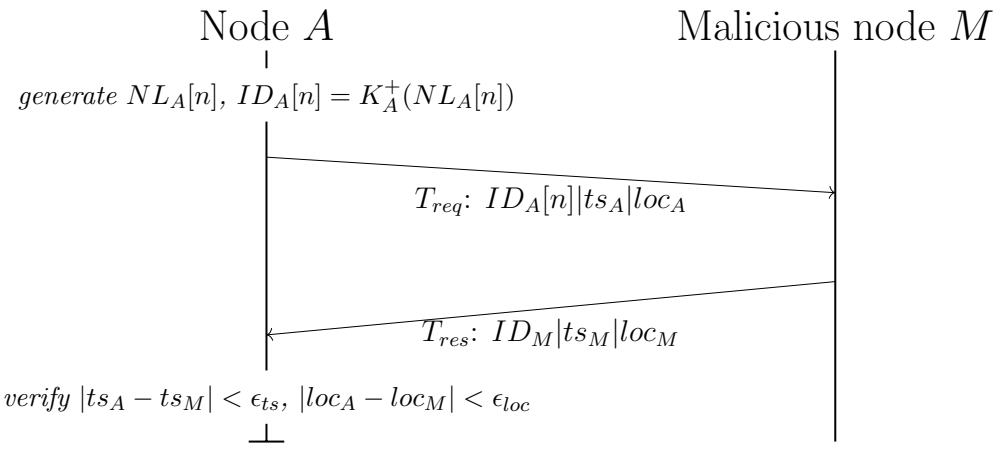


Figure 3.5: An aborted transaction due to malicious node M

To abort the exchange, node A terminates the ad-hoc network with node M, and won't publish anything onto the blockchain. This means that even if node M fabricates some data from node A and publishes a transaction onto the blockchain, the matching transaction from node A will not be present, so node M's transaction will be disregarded by any Verifiers.

3.3 Verification

In the verification stage, a mobile node attempts to prove its location to a Verifier node, e.g. a bank. To do this, the mobile node sends the Verifier a number of parameters, as shown in Figure 3.6.

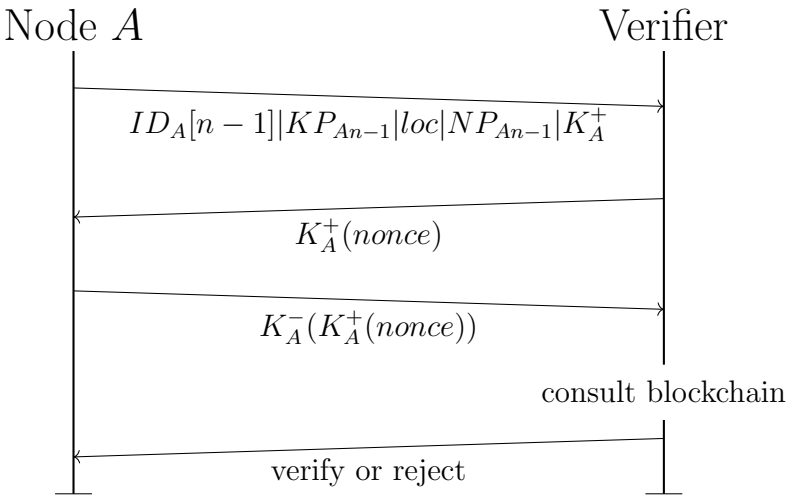


Figure 3.6: Verification request

$ID_A[n-1]$ is the ID of node A's most recently published transaction. Note that $ID_A[n]$ has already been generated, but hasn't been used in a transaction yet. Node A has generated it so that it is ready to be used in the next transaction.
KP_{An-1} is node A's key packet up to transaction $n-1$, and includes keys for as many transactions as node A feels is appropriate in order to receive verification.
loc is node A's current claimed GPS coordinates for which it is seeking verification.
NP_{An-1} is node A's *Nonce Packet*, a subset of its *Nonce List* (section 3.1.1). A Nonce Packet is the list of the nonces used to generate node A's most recent IDs, up to $n-1$, that node A is willing to share with the Verifier.

The goal of the verification stage is for the Verifier to either accept or reject the location that the user is claiming. Given $ID_A[n-1]$ and KP_{An-1},

a Verifier is able to retrieve node A's previous transactions, and the transactions of all of its alibis. This will allow the Verifier to recursively inspect the transactions of all alibis used in a node's transactions, until it is satisfied that the location can be verified or rejected. This recursive inspection can be visualised as a tree, as shown in figure 3.7.

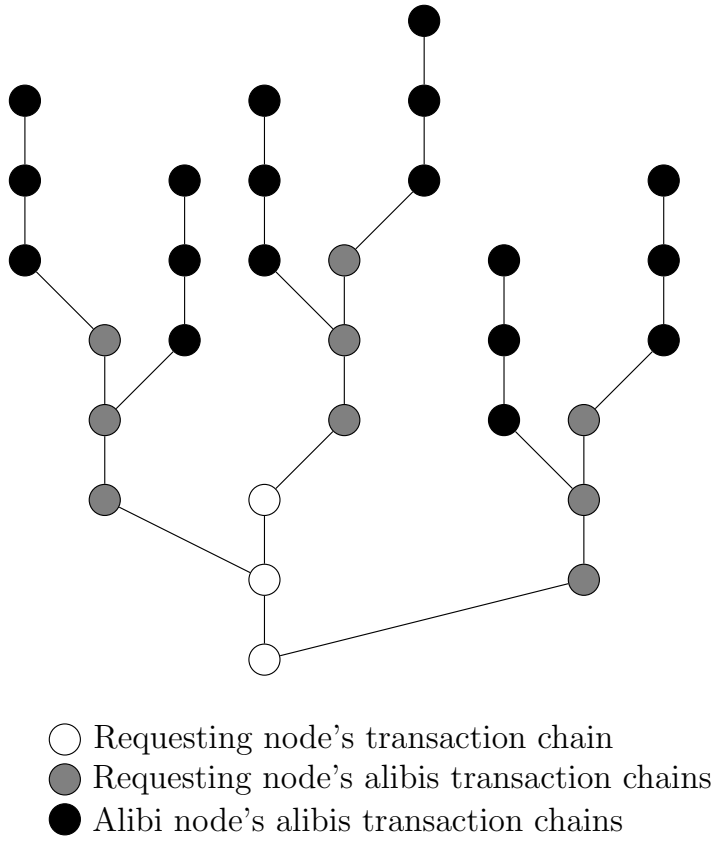


Figure 3.7: Alibi chain relationship tree

If the verifier wishes to recurse more, it could inspect the black node's alibis' transaction chains. Note that the example in figure 3.7 is a simple one, with $n = 3$ being the length of the transaction chain subset provided for verification. In reality n should be much larger.

There are a huge number of possible factors involved in reaching a conclusion on the verifiability of a user from the data in the blockchain; some of

these factors will likely be unique to certain Verifiers, and kept secret to improve reliability and avoid gaming by malicious users. Some simple example factors include:

- **Alibi credibility:** if each of the node's alibis have little or no alibis themselves, they will not be considered credible alibis, and a Verifier will reject the verification request (see figure 4.1).
- **Transaction relevance:** if a Verifier considers the transactions it receives to be out of date, the request will be rejected on the grounds that the transactions are no longer relevant.
- **Alibi reuse:** if a Verifier can prove that alibis are being reused frequently, then it will reject the verification request. Due to transaction anonymity, a Verifier may not always be able to tell if two alibis are the same person or not. e.g. node A could use node B as an alibi, then wait for node B to complete n more transactions, then use node B again. This is not a viable attack on the verification system. Depending on the size of n, enough time may pass between both uses of node B as an alibi that the transactions are no longer relevant.

Chapter 4

Evaluation

The previous chapter introduced a model for a decentralised location proof system. This chapter will present an evaluation of the model with respect to the design goals defined in section 2.5.

4.1 OTIT conformance

As explained in section 2.5.2, the OTIT model [4] defines 8 desirable properties of a location proof system. The model introduced in chapter 3 is evaluated with respect to OTIT below.

4.1.1 Chronological

The chronological property of OTIT states that location proofs should be ordered according to the sequence of their creation.

This property is not enforced at the time of transaction publication in this system. This is to avoid revealing any transaction information to the miner nodes, therefore preserving the privacy of the transaction owner. Instead, the Chronological property is enforced at the time of verification. The chronological link created by adding $KL_{AT}(ID_A[n-1]|ts_A)$ to the published transaction allows Verifier nodes to prove that the transaction is chronologically ordered. Therefore the system preserves OTIT's "Chronological" property.

4.1.2 Order preserving

This property states that the order in which location proofs are entered into the proof chain should be maintained.

In this system, a blockchain is used to store location proof transactions. In a blockchain, new blocks are added in a linear, chronological order, and cannot be modified so long as the majority of miners in the network are honest [11].

A block consists of a number of transactions. These transactions are included in the block in the order they were received by the miner who solved the proof-of-work for that block. Once the block has been signed into the blockchain, it is impossible for the transactions inside the block to be reordered, assuming the miner network is controlled by a majority of honest nodes.

As a further method of preserving order, each node includes its observed timestamp in its encrypted transaction and link data. This prevents a group of malicious miner nodes from altering the order and timestamp of the transaction before signing it into the blockchain. Therefore the transactions stored in these blocks satisfy the "Order preserving" property of OTIT.

4.1.3 Verifiable

This states that a Verifier should be able to verify or reject the claim by the user that it has visited the given location(s).

The sytem described in chapter 3 inherently satisfies this property of OTIT. Verifier nodes may be operated by banks, employers, or other parties who have a vested interest in allowing their customers or employees to be able to prove their location. These Verifiers can verify or reject a mobile nodes claimed location by examining the nodes chronological link of transactions (as shown in figure 3.7). The Verifier will recursively generate and examine this tree until it can conclude whether or not the transaction chain is valid.

4.1.4 Tamper evident

The system is tamper evident, meaning that a Verifier is able to detect if the proof chain has been tampered with or modified. If a malicious node creates a successful transaction with an honest node, but modifies the transaction data before publishing, a Verifier node will reject the malicious transaction. This

rejection will be on the grounds that the matching transaction, published by the honest node, contains contradictory data.

The design assumes the underlying security of the blockchain, so any malicious miner nodes will not be able to tamper with existing transaction data providing that the majority of CPU power in the miner network is owned by honest miners [12]. Therefore the tamper evident property is satisfied.

4.1.5 Privacy preserved

This property states that when the proof chain is verified for specific location proofs, privacy must be preserved for the other location proofs within the chain.

In the system, privacy is preserved using *Key Lists* (See section 3.2.1). Two keys are used for each transaction; $KL_{AT}[n]$ is used to encrypt the *transaction data* for node A's nth transaction, and $KL_{AL}[n]$ is used to encrypt the *backwards chronological link* between node A's nth transaction and transaction $n-1$.

To control its privacy, a user can choose only to reveal $KL_{AL}[n]$ for a specific transaction n. This may be useful if a user does not wish to reveal transaction data for transactions created within a certain distance of its house, for example. Hiding $KL_{AT}[n]$ while revealing $KL_{AL}[n]$ has the advantage of preserving the user's privacy while maintaining a clear backwards chronological link of transactions for a Verifier to follow. This simultaneously preserves both privacy and chronological integrity.

4.1.6 Selective in-sequence privacy

This is the requirement that the user must be able to choose not to reveal certain location proofs within the proof chain when seeking verification. The system satisfies this property by using *Key Packets* (section 3.2.4).

A *sub-set* of size m of a node A's proof chain, beginning at proof n, contains all proofs in the sequence $ID_A[n-m]$ to $ID_A[n]$. Every transaction in this chain is split into two parts, a transaction data and link data part, and separately encrypted. If a node reveals keys for a Verifier to decrypt all transactions in the sequence $ID_A[n-m]$ to $ID_A[n]$, it will not be possible for the Verifier to discover transaction $ID_A[n-m-1]$, or any transaction outside the sequence $ID_A[n-m]$ to $ID_A[n]$.

By encrypting each transaction with a different pair of keys, this system satisfies OTIT's "selective in-sequence privacy" property.

4.1.7 Privacy protected chronology

This property of OTIT states that the location proof system should ensure that the user does not hide important information from the items within the subset of the proof chain. In this system, it is impossible by design to hide important information, due to the use of backwards chronological linking between transactions.

4.1.8 Convenience and derivability

This requirement states that the system must ensure that the user does not burden the Verifier with a large amount of data to analyse in order to verify the user's location.

In this system, the data is stored in a central blockchain that the Verifier has access to, and the user provides the Verifier with decryption keys and indexes into just a sub-section of this data. Therefore the data sent to from the user to the Verifier, KP_{An}, is quite small in size.

The Verifier has the freedom to decide how comprehensively to investigate the users data; It can decide how far to recurse while investigating the tree of alibi relationships. This property of OTIT is therefore satisfied by allowing the Verifier to decide how complex the verification process needs to be.

4.2 Threat analysis

4.2.1 False presence

In a false presence attack, a dishonest user attempts to obtain a location proof for a false location.

In this system, any dishonest user attempting to obtain a false location proof with an honest alibi will not succeed, as the honest alibi will abort the exchange (see section 3.2.5). A dishonest user who publishes a transaction onto the blockchain with a fabricated alibi will also not succeed, as the transaction will be rejected during verification if a Verifier can't find the matching

alibi transaction. The system is therefore not vulnerable to a false presence attack.

4.2.2 Malicious intruders

A malicious intruder offers to help another user to prove a false location, but doesn't prove a false location for itself.

In the context of this system, a malicious intruder may collude with another user to prove a false location, by acting as an alibi. These transactions will be successfully published onto the blockchain. However, if this user attempts to have its transaction chain verified, a Verifier will quickly notice the lack of credible alibis in the proof chain, indicating malicious activity, and reject the proof request. A sample tree of a node with uncredible alibis is shown in figure 4.1. This avoids a malicious intruder vulnerability.

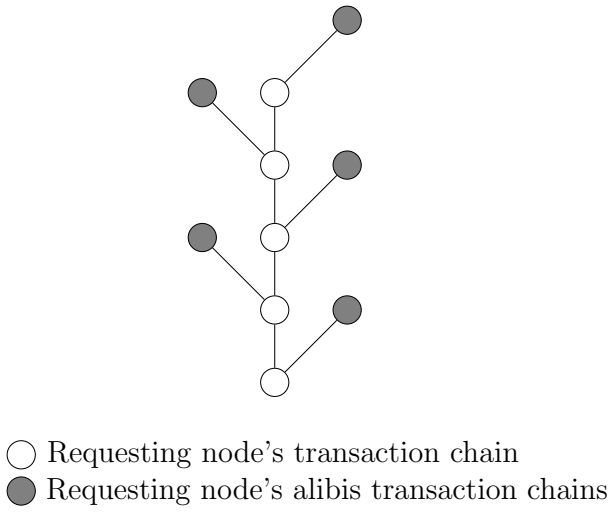


Figure 4.1: A proof chain with uncredible alibis

4.2.3 Curious users

A curious user is one who learns another users identity by monitoring its location proofs.

Identity discovery is impossible by design in this system. By using a different, seemingly random identity for each transaction, it is impossible for a curious user to determine if two transactions belong to the same user or not. It is therefore impossible for a curious user to learn another user's identity by monitoring location proofs.

4.2.4 Malicious applications

A malicious application is one that tries to take advantage of a user's information as the user seeks verification.

This attack does not affect this system, as a user will only seek verification from applications (Verifiers) it deems trustworthy. Verifiers, such as banks or employers, will be known to the user, and therefore trusted. The user also has complete control over which Verifiers it chooses to seek verification from, it is not obliged to send verification requests to any Verifier.

4.2.5 Eavesdroppers

An eavesdropper intercepts or modifies communication between two honest users. This threat does not affect this system, as ad-hoc bluetooth networks are set up between two mobile nodes advertising their current transaction ID, e.g. $ID_A[n]$. An eavesdropper will not be a member of this ad-hoc network, and therefore cannot modify any communication in the network.

4.2.6 Wormhole attacks

In a wormhole attack, an attacker records network traffic in one location and replays it in another location, at another time.

This attack is not relevant to the system presented in chapter 3, as this system is decentralised. Any wormhole attack transmissions will simply be ignored by the intended victim.

4.2.7 Weak identities

In a weak identity attack, a user gives its public key or identity to another colluding user in order to create a location proof.

A weak identity attack will not affect this system. Exchanging public keys or identities between colluding users offers no advantage. This is because a

Verifier will ensure that any users requesting verification own the private key corresponding to the public key transferred during the verification request (see section 3.3).

4.2.8 False timestamping

False timestamping includes backdating and future dating attacks. In a backdating attack, the attacker creates a location proof with a past timestamp. In future dating, the attacker creates a location proof with a future timestamp.

This system is resilient against a false timestamping attack by design. Every transaction is created with a reference to an alibi transaction. If an attacker creates a transaction with a past or future timestamp, the corresponding alibi transaction will contain contradictory information, and a Verifier will disregard it.

4.2.9 Implication

An implication attack occurs when a malicious user, or group of colluding malicious users, create a false location proof for an honest user.

This system is resilient to an implication attack. Location proofs are stored in a public blockchain, but the decryption keys, nonce and public key used to create the transaction all reside with the user who created it. A user cannot be implicated in a location proof without creating the identity used in the proof, and encrypting the transaction data.

It is therefore impossible to implicate an honest user in a location proof, without exploiting vulnerabilities outside the location proof system (for example by compromising the underlying assumed security of the device).

4.2.10 False assertion

In a false assertion attack, two malicious users collude to create false location proofs for each other.

It is possible in this system for two malicious users to collude as alibis and publish their transactions on the blockchain. This is allowed because the miner nodes do not know the contents of the data they are signing into the blockchain, as it is encrypted.

Providing that the system is comprised of a majority of honest nodes, this attack will not succeed. It will be obvious to a Verifier after constructing a

graph of the node's alibi transaction (as shown in section 3.3) that the graph lacks alibi credibility, and therefore two malicious users are likely colluding to create location proofs for themselves. Therefore the system is not vulnerable to this attack.

4.2.11 Proof switching

In a proof switching attack, a malicious user modifies one of its existing honest proofs to spoof a false location.

This is not possible in this system, as the blockchain is used as a tamper-proof method of storing location proofs. Once the proof is included in the blockchain, it cannot be modified.

If the proof has not been added to the blockchain yet, then it is possible for the malicious user to modify it to attempt to spoof a false location. However this attack will not work on this system. Any Verifier will reject the transaction on the grounds that it contains contradictory information to the alibis transaction, which has not been modified.

4.2.12 Relay attack

In a relay attack, a malicious user uses a colluding *proxy user* physically present at the location that the malicious user wishes to create a false proof, to create a location proof with an honest user.

This system is resilient against most relay attacks. This is because proofs created using a proxy user will be inconsistent with the other proofs in a users proof chain. For example, a user living in New York uses a proxy user to create a proof with a user in London. A Verifier will reject the users claim that it is in London, because of the lack of alibi credibility.

One vulnerability of the system can be called a *complete* relay attack. In a *complete* relay attack, two malicious nodes exist purely for the purpose of proxying location proof. One node, the proxy, will *never* request location proofs for itself. Its only purpose is to proxy exchange data between honest nodes and the other malicious node, the *attacker*. The attacker will never request a proof by itself, it will always route its requests through the proxy user.

Resilience against a complete relay attack scenario must be taken out of the scope of this system.

4.2.13 Sybil attack

A Sybil attack occurs when a single malicious user generates multiple *pseudoidentities*, and begins masquerading as multiple users.

Previous work has shown that preventing a Sybil attack is impossible in a fully decentralised system [5]. This system is therefore vulnerable to a highly targeted Sybil attack. However, there are various possible methods of mitigating against such an attack.

Introducing a penalty for identity creation would mitigate against an attempted large-scale Sybil attack. For example, if a node is required to solve a proof-of-work puzzle before creating an identity, a large-scale Sybil attack would prove extremely computationally expensive for the attacker. This is an impractical mitigation however, as this system is designed to operate on mobile phones. Mobile phones should not be forced to complete proof-of-work puzzles, as computational resources and power supply is very limited when compared with miner nodes, which are servers.

Another possible mitigation involves using a web of trust, which allows users of a system to express their trust of other users [15]. Users of the system build a large network of trust relationships, allowing other users to assess the trustworthiness of a potential alibi before creating an exchange with it. This technique of mitigating against the Sybil attack would involve a trade-off with the privacy protecting features currently present in the system, because a web of trust requires globally unique, static user identification to operate. This system uses non-unique, dynamic user identification in order to preserve privacy.

Previous research [5] has concluded that the only provably correct way of preventing a Sybil attack in a decentralised system is to use a *centralised* name system to coordinate the distribution of identities. Therefore a Sybil attack vulnerability must be taken out of the scope of this system, as this system is focused on a decentralised approach to location verification.

Chapter 5

Conclusion and Future Work

5.1 Future work

There are many worthwhile areas for future work, because to the best of the author's knowledge, this is the first attempted design of a fully decentralised location proof system.

5.1.1 Verification techniques

Section 3.3 outlined a number of simple verification techniques that a Verifier node could use to analyse the a proof chain. There is potential for further research on advanced techniques of analysing proof chain trees to determine the validity of a user's proof request.

5.1.2 Privacy and verifiability

As explained in section 3.2.4, certain transactions can be hidden from a proof chain in order to protect a user's privacy, but maintain chronological integrity. There is an interesting area of further work in studying the effect that hiding certain proofs has on that user's verifiability.

5.1.3 Sybil attack

Previous work has indicated that there is no provably correct way of preventing a Sybil attack in a decentralised system [5]. However, further work studying strong mitigations against a Sybil attack, or even preventing a Sybil

attack entirely, would greatly improve the appeal of a decentralised location proof system, and potentially lead to its ubiquitous use.

5.2 Conclusion

This project presented the design of a decentralised location proof system. The aim of the project was to design a system that satisfied the design goals, as defined in section 2.5. This system satisfies all of the requirements of OTIT [4], and all but one of the threats defined in section 2.5.3. However, chapter 4 concluded that this Sybil attack threat must be highly targeted to be effective.

In conclusion, the model presented in this project is a viable design of a decentralised location proof system, providing that the Sybil attack vulnerability can be heavily mitigated against or prevented outright.

Bibliography

[1] J. Brassil, P.K. Manadhata, "Verifying the Location of a Mobile Device User", Proc. of MobiSec 2012, June 2012.

[2] W. Luo, U. Hengartner, "Proving your Location without giving up your Privacy", Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications, HotMobile 2010, Annapolis, Maryland, February 22 - 23, pp. 7–12. ACM, New York (2010).

[3] R. Khan, S. Zawoad, M. Haque, and R. Hasan, "'Who, When, and Where?' Location Proof Assertion for Mobile Devices", Proceedings of the 28th Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy, ser. DBSec. IFIP, July 2014.

[4] R. Khan, S. Zawoad, M. Haque, and R, Hasan. "OTIT: Towards secure provenance modeling for location proofs", Proc. of ASIACCS. ACM (2014).

[5] J. Douceur, "The Sybil Attack", In First IPTPS, March 2002.

[6] MaxMind LLC. GeoIP. http://www.maxmind.com, 2010.

[7] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida, "Constraint-based geolocation of Internet hosts", IEEE/ACM Transactions on Networking, 14(6):1219–1232, 2006.

[8] N. O. Tippenhauer, K. B. Rasmussen, C. Popper, and S. Capkun, "iPhone and iPod location spoofing: Attacks on public WLAN-based positioning systems", SysSec Technical Report, ETH Zurich, April, 2008.

[9] R. Dingledine, N. Mathewson, and P. Syverson, "TOR: The second generation onion router", Proceedings of the Usenix Security Symposium, 2004.

[10] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, "Distributed Systems: Concepts and Design (5th Edition)", Boston: Addison-Wesley, 2011.

[11] M. Swan, "Blockchain: Blueprint for a New Economy", Sebastopol, CA, O'Reilly Media, 2015

[12] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008

[13] V. Chandrasekhar, J. Andrews, and A. Gatherer, "Femtocell Networks: A Survey", IEEE Communications Magazine, Vol. 46, No. 9, pps. 59-67, September 2008

[14] J. Mirkovic, P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms", ACM SIGCOMM Comput. Commun. Rev. 34, 2, 39–53, 2004

[15] R. V. Guha, R. Kumar, P. Raghavan, and A. Tomkins, "Propagation of trust and distrust", in Proceedings of WWW, pp. 403–412, 2004.