Decentralised location proof system

Conor Taylor

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

Contents

List of Figures

Chapter 1

Introduction

1.1 Project and Motivation

Location verification is the process of verifying whether or not a node (computer) is physically present at the location it claims to be. Existing location verification systems attept to provide this service using centralised, trusted "authoritive" nodes to provide proof of another node's location. These approaches require investment in infrastructure, and are subject to attacks on privacy and denial of service attacks. This project aims to present a *decentralised* solution to this problem, in which there is no "authoritive source" trusted and relied upon to provide and store sensitive location information.

This project describes a decentralised location proof system that operates on mobile devices (mobile nodes). I propose a design in which location proofs are obtained by using other untrusted mobile nodes as *alibi's*. Proofs will be created by communicating over an ad-hoc bluetooth network, encrypted, and published 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.

Chapter 2

Background

2.1 Proving your 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 proving is a large part of that security. There is currently no ubiquitous method of *proving* your location digitally. Many companies, such as Netflix, use GeoIP databases such as MaxMind [6] to determine a users location based on his IP address. This technique is inaccurate and easily bypassable using proxy servers or VPNs.

One newer, more advanced tecnique of GeoIP location verification is Constraint-Based Geolocation [7]. This is an active geolocation technique, compared with GeoIP verification which is static. It operates by estimating the location of the 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* are systems which allow users of the system to *prove* their location to another user. Examples of these 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 computers (nodes) connected over a network. These nodes then communicate to complete the task by passing messages 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 two main advantages; It's far more efficient, because different index calculations can be computed on different nodes at the same time, and it's very scalable, because nodes can be added or removed from the network dynamically, to react to 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 don't act as slaves to a master. Bitcoin [12] is a good example of a large-scale decentralised system. Two nodes in the Bitcoin network can choose to create a transaction by sending money between themselves, and they make the details of that transaction public. The two nodes created and published the transaction independently of all other nodes on the network.

In the context of a location proof system, I consider a distributed system to be one 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, while maintaining centralised control over the system. This is shown in figure 2.1.

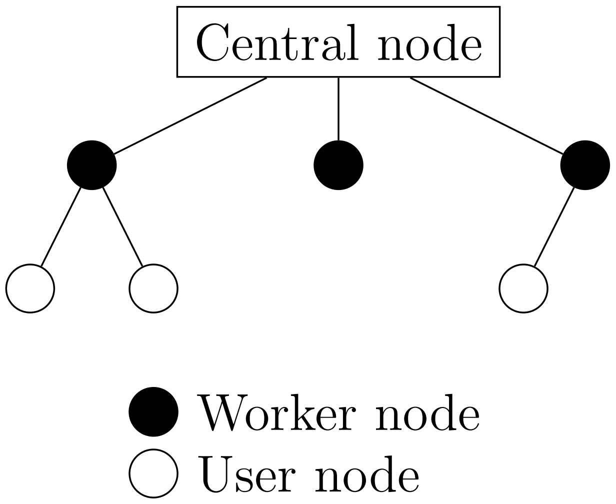

Figure 2.1: Distributed location proof system

I consider a decentralised location proof system 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 the interaction (figure 2.2).

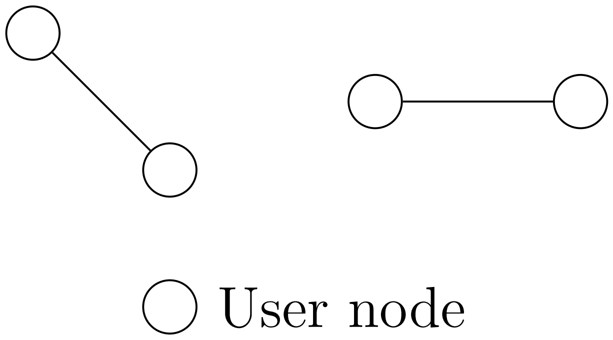

Figure 2.2: Decentralised location proof system

2.3 Existing location proof systems

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

A hardware technique [1], developed by Brassil et al. of HP Laboratories, operates by supplementing existing WiFi access points (*AP's*) with *femtocells*. A femtocell is a small cellular antenna that connects to a mobile carrier via the Internet. Location verification over the internet is made possible by determining which femtocell a mobile node is connected to as it transfers data via 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 WiFi access points, and requires access to mobile providers' user database to identify users locations (see figure 2.3).

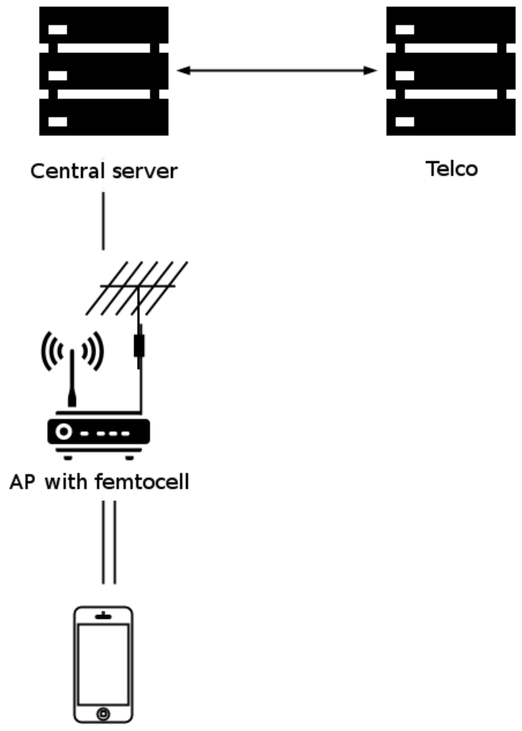

Figure 2.3: Hardware-based proof system (Adapted from Brassil et al. [1])

The use of a centralised system, as described above, creates security, privacy and vulnerability issues. An attacker who succeeds in compromising the security of the central server can violate the privacy of the users of the system, and potentially track their location. This is because in this system, location proofs reside with the central server, so the user has no control over their security. The central system architecture is also vulnerable, in the sense that a resource availability attack such as a DDoS attack 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).

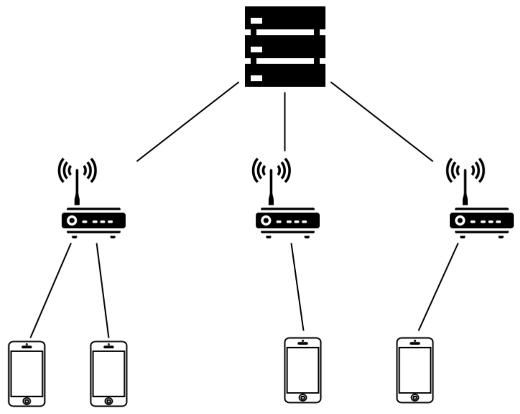

Figure 2.4: Access point proof system (Adapted from Luo et al. [2])

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 **??**. This can then be submitted to a Verifier.

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) he 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 2.5). 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.

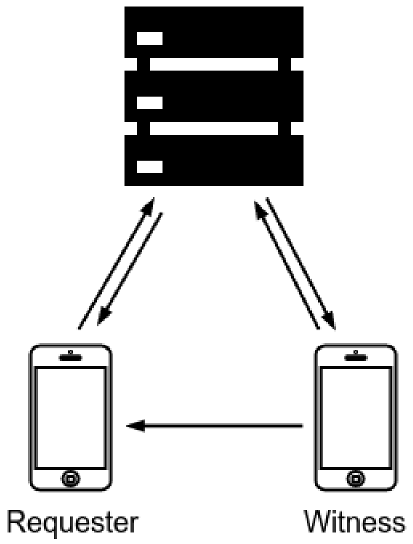

Figure 2.5: Proof system using witnesses (Adapted from Khan et al. [3])

2.4 Ad-hoc networks

2.5 Blockchain

2.6 Goals of a decentralised proof system

Chapter 3

Design

The location proof system we propose allows *mobile nodes* to create location proofs for each other after physically meeting. When two mobile nodes are in close physical proximity, they initiate a transaction over a short range, ad-hoc network such as Bluetooth.

To create a transaction, two mobile nodes anonymously share their GPS coordinates and current time over a short-range network. Both nodes then check that these parameters don't differ by more than some value ϵ. Once satisfied, they each create an encrypted, privacy-protecting transaction logging their location and the identity of the other mobile node (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 (by transferring decryption keys), a *Verifier node* can consult the data in blockchain and determine whether or not the mobile node is present at its claimed location. The blockchain is managed by a network of *Miner nodes*, who collect transactions from mobile nodes, distribute them across the network, and try to create a new *block* of transactions by attempting to solve a *proof-of-work* puzzle.

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 transaction. This means that when the mobile node seeks verification from a Verifier node, the Verifier can examine the mobile node's alibi's, and each of the alibi's recent alibi's, forming a large graph of connections which can be used to verify or reject the mobile nodes claimed location. 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.

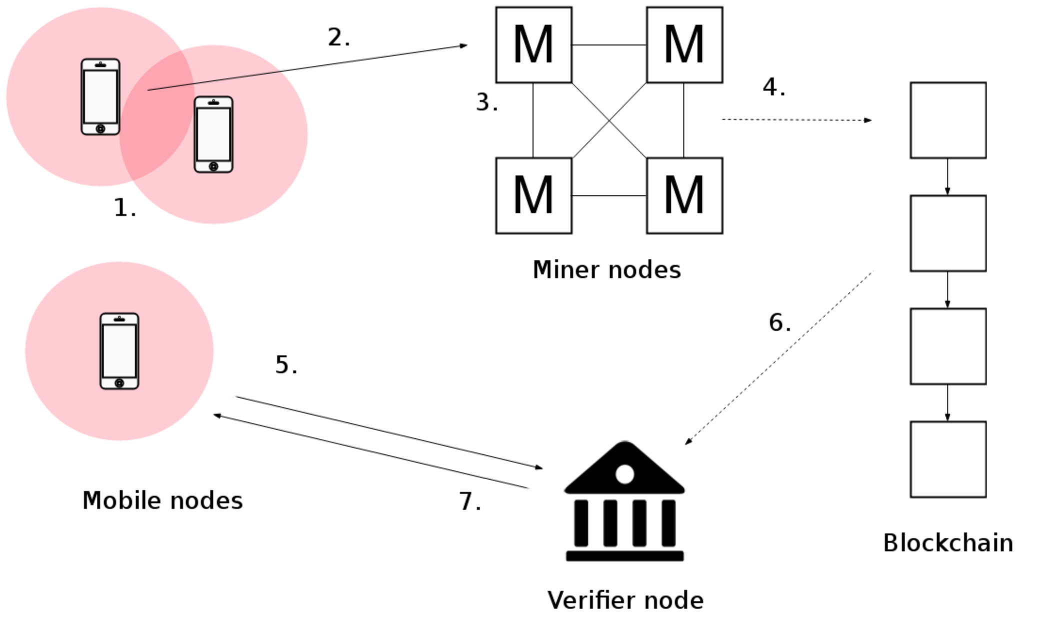

Figure 3.1: Design and operation overview

1. Two mobile nodes in close physical proximity create a transaction.

2. The mobile nodes each publish their transaction to a Miner node.

3. The Miner node distributes the transaction across the Miner network.

4. One Miner node solves the proof-of-work for a block of transactions, and the new block of transactions 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 uses 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 transactions in the blockchain.

3.1 Identities

In order to preserve user's 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 the 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 is created, 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, the node will provide the Verifier with its public key, along with the list of the nonces used to generate its n most recent transactions (see figure 3.4). This allows the Verifier to calculate all of the node's identities, and prove that they were all created by the same node, and can then retrieve the transactions associated with those identities from the blockchain. The Verifier will also ensure that the node owns the private key associated with the provided public key, to prevent identity theft.

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 generates a new identity, it must first generate a new 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).

3.1.2 Identity duplication

Since all identities are generated from random nonces, there is nothing to stop 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 in the blockchain. The Verifier will have a decryption key for the transaction it is searching for, which will only be able to decrypt one of the transactions with the duplicate ID. The Verifier will try to decrypt each one until it finds one that can be decrypted successfully.

3.2 Transactions

During a transaction between two nodes, a number of different items must be shared and calculated. Figure 3.2 describes an honest, successful transaction, assuming an ad-hoc Bluetooth network has already been set up between nodes A and B.

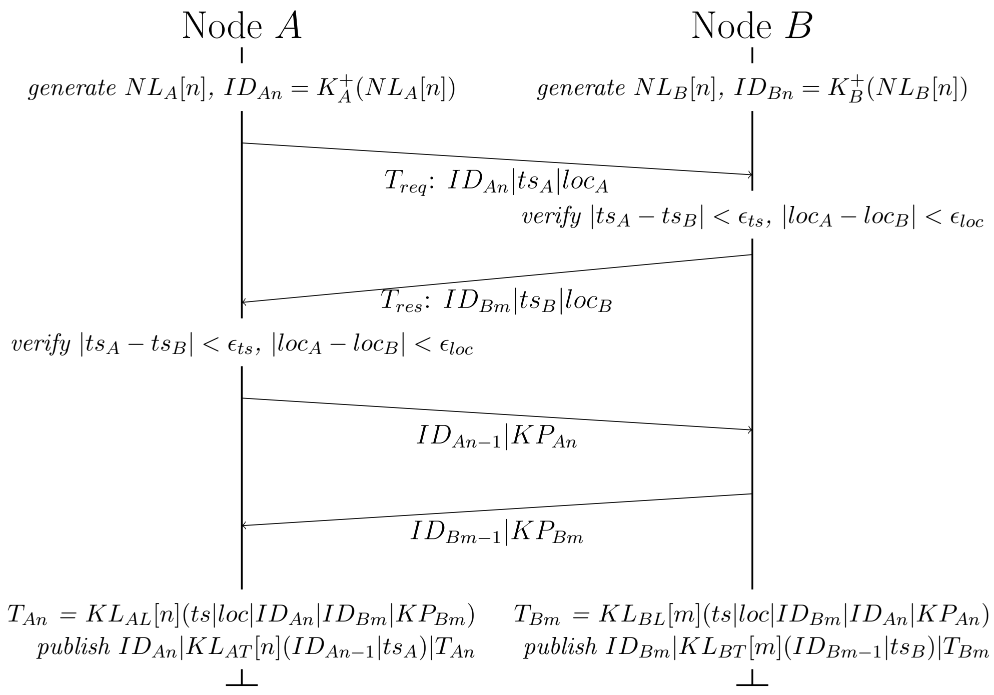
Node A
Node B
generate $NL_A[n]$, $ID_{An} = K_A^+(NL_A[n])$
generate $NL_B[n]$, $ID_{Bn} = K_B^+(NL_B[n])$
T_{req}: $ID_{An}|ts_A|loc_A$
verify $|ts_A - ts_B| < \epsilon_{ts}$, $|loc_A - loc_B| < \epsilon_{loc}$
T_{res}: $ID_{Bm}|ts_B|loc_B$
verify $|ts_A - ts_B| < \epsilon_{ts}$, $|loc_A - loc_B| < \epsilon_{loc}$
$ID_{An-1}|KP_{An}$
$ID_{Bm-1}|KP_{Bm}$
$T_{An} = KL_{AL}[n](ts|loc|ID_{An}|ID_{Bm}|KP_{Bm})$
publish $ID_{An}|KL_{AT}[n](ID_{An-1}|ts_A)|T_{An}$
$T_{Bm} = KL_{BL}[m](ts|loc|ID_{Bm}|ID_{An}|KP_{An})$
publish $ID_{Bm}|KL_{BT}[m](ID_{Bm-1}|ts_B)|T_{Bm}$

Figure 3.2: Honest, successful transaction

Where:

NL$_\mathbf{A}$[n] is the nth nonce in A's *Nonce List*, which is used to generate ID_{An} (see section 3.1.1).

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* and are assumed to have been initialised and populated before the transaction described in Figure 3.2 begins.

After creating an ad-hoc Bluetooth network between nodes A and B, the first step in the transaction is for both nodes to share their IDs, and agree upon a GPS location and current time. Nodes likely won't share the exact same GPS coordinates or time, but once the values don't differ by more than some small constant ϵ, the nodes will continue with the transaction. If node A claims a GPS location or timestamp that differs from what node B senses by more than ϵ, node B will abort the transaction.

After nodes agree upon the parameters of the transaction, they exchange their previous ID, along with their *Key Packets*. A *Key Packet* is a list of keys used in a node's n most recent transactions, along with the transaction ID's if necessary. This allows node A to decrypt node B's m most recent transactions, therefore providing an alibi for node A.

3.2.1 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 object, T, to publish onto the public blockchain. The following transaction data will be calculated by node A, after communicating with node B:

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

Where:

$\mathbf{n}$ is node A's current transaction number (i.e. A has published $n-1$ transactions before now).
$\mathbf{KL_{AT}}$ is one of A's *Key Lists*, used to encrypt transactions.
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.
$\mathbf{ID_{An}}$ is A's *nth* identity, used for this transaction only.
$\mathbf{ID_{Bm}}$ 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 3.2.3

3.2.2 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 the transaction.

Backwards-chaining is an important part of verifiability. The Verifier must be able to prove that the proof chain provided by the mobile node has not been chronologically re-ordered. The transaction must therefore be accompanied by a link to the node's chronologically previous transaction. This link must be encrypted, along with the transaction timestamp, to prevent a curious user from monitoring the public blockchain to observe a user's transaction publishing activity. A's copy of the transaction log P_{An} will therefore be in the form:

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

To publish P_{An}, A must send P_{An} to a miner node, who will add it to its *pending transaction* queue, and send it to other miners in the network to add it to their queues as well. The transaction will then be signed into the next block in the blockchain.

3.2.3 Key Packets

The 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 he doesn't 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 doesn't want to reveal transaction T_{Bm-1}, he would reveal 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 may be needed, for example, if node B doesn't want to allow anyone to decrypt any transactions he has created withing 5km of his home, in order to protect his privacy.

3.2.4 Aborting transactions

A node may decide to abort a transaction if it thinks it is in contact with a malicious node. In figure 3.3, node A encounters a malicious node M, who tries to spoof its identity with A. Node A will abort the transaction once it notices that $|ts_A - ts_M| \geq \epsilon_{ts}$, or $|loc_A - loc_M| \geq \epsilon_{loc}$.

To abort the transaction, A terminates the ad-hoc network with node M, and won't publish anything onto the blockchain. This means that even if M fabricates some data from A and publishes a transaction onto the blockchain, the matching transaction from A will not be present. Any transaction on the blockchain that do not contain a reference to a valid alibi transaction will be disregarded during verification.

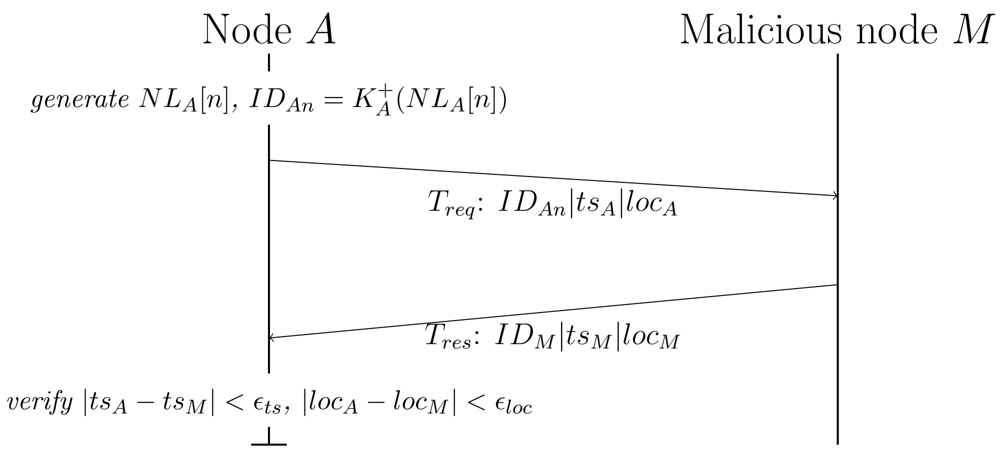
Node A
Malicious node M
generate $NL_A[n]$, $ID_{An} = K_A^+(NL_A[n])$
T_{req}: $ID_{An}|ts_A|loc_A$
T_{res}: $ID_M|ts_M|loc_M$
verify $|ts_A - ts_M| < \epsilon_{ts}$, $|loc_A - loc_M| < \epsilon_{loc}$

Figure 3.3: An aborted transaction due to malicious node M

3.3 Verification

In the verification stage, a mobile node tries 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 3.4:

$\mathbf{ID_{An-1}}$ is the ID of A's most recently published transaction.

$\mathbf{KP_{An-1}}$ is A's key packet up to transaction $n-1$, and includes keys for as many transactions as A feels is appropriate in order to receive verification.

loc is A's current claimed GPS coordinates for which it is seeking verification.

$\mathbf{NP_{An-1}}$ is 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 A's most recent IDs, up to $n-1$., that A is willing to share with the Verifier.

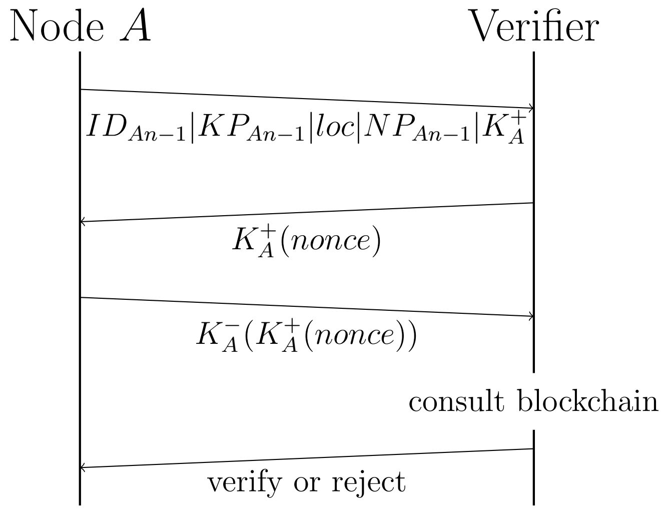

Figure 3.4: Verification request

The goal of the verification stage is for the Verifier to either accept or reject the location that the user is claiming. Given ID_{An-1} and KP_{An-1}, a Verifier is able to retrieve A's previous transactions, and the transactions of all of its alibi's.

The Verifier uses NP_{An-1} and K_A^+ to prove that node A was the author of all of the transactions he claims to be. This prevents a collusion attack whereby nodes could share common transactions between different proof chains. When walking chronologically backwards along A's proof chain, the 3rd party will ensure that the ID used in transaction m on the proof chain matches $K_A^+(NP_{An-1}[m])$.

There are a huge number of possible factors involved in reaching a conclusion from the data on the blockchain; some of these factors will likely be unique to certain Verifiers, and kept secret to improve reliability and avoid gaming. Some simple example factors may include:

- **Alibi credibility:** if each of the node's alibi's have little or no alibi's themselves, then the node is likely attempting a poorly-constructed Sybil attack, and the Verifier will reject his verification request.
- **Alibi reuse:** if the Verifier can prove that alibi's are being reused frequently, then it will reject the verification request.
    - Due to transaction anonymity, the Verifier may not always be able to tell if two alibi's are the same person or not. e.g. node A could use node B as an alibi, then wait for 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

In this paper, we have presented a model for a location proof system which aims to preserve privacy and prevent false proofs from being created.

4.1 OTIT conformance

We can compare our model to OTIT [4], a list of features and requirements necessary for designing any secure location proof scheme. OTIT presents 8 such requirements:

> **Chronological**: The chronological property of OTIT ensures that transactions are published in the order they were created. In our system we do not enforce this property at the time of transaction publication. This is to avoid revealing any transaction information to the miner nodes, therefore preserving the privacy of the transaction owner. Instead, we enforce this property at the time of verification. The chronological link created by adding $KL_{AT}(ID_{A-1}|ts_A)$ to the transaction will allow Verifier nodes to prove that the transaction is chronologically ordered, preserving OTIT's "Chronological" property.
>
> **Order preserving**: This property states that the order of transactions cannot be modified after they are created. In our system, a blockchain is used to store 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]. Therefore the transactions stored in these blocks satisfy the "Order preserving" property

of OTIT.

Verifiable: This states that the proof, and the order or proofs, should be verifiable by a trusted auditor. In our case, a Verifier node is an auditor trusted by the mobile node seeking verification. A Verifier node may be operated by a bank, government, or any entity wishing to make use of our location proof system.

Our system inherently satisfies this property of OTIT, as Verifier nodes are required to verify or reject a mobile node's claimed location. This is done by examining the mobile node's chronological link of transactions, and recursing for each alibi used for each transaction along the chronological link until the Verifier can conclude if the transaction chain is valid or not.

Tamper evident: Our system is tamper evident. 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.

We assume 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 mining system is owned by honest miners [12]. Therefore the tamper evident property is satisfied.

Privacy preserved: This states that the user should have control over his level of privacy exposure when using the system. In our system, this is provided using *Key Lists* 3.2. We provide two keys for each transaction; $KL_{AT}[n]$ is used to encrypt the *transaction data* for A's *nth* transaction, and $KL_{AL}[n]$ is used to encrypt the *backwards chronological link* between A's *nth* transaction and transaction $n-1$.

To control his 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 his 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 the Verifier to follow, preserving both privacy and system consistency.

Selective in-sequence privacy: This is the requirement that any proof chain must support *sub-set verification*, allowing a user to provide only a sub-set of proofs from his current proof chain for verification. Our system satisfies this property by using *Key Packets* (section 3.2.3).

In our system, a *sub-set* of A's proof chain can be defined as KP_{An}. The sub-set therefore begins with ID_{An}, and ends when there is no key $KL_{AL}[n-m]$ in KP_{An} to decrypt the link to transaction T_{An-m-1}.

Privacy protected chronology: This states that the proof system, which provides *selective in-sequence privacy*, should also ensure that the user does not "hide away" important items within the subset. This is impossible in our system due to the use of backwards chronological linking between transactions, so our model satisfies this property.

Convenience and derivability: This requirement states that the verification process should be convenient, and the user should not burden the Verifier node with a huge load of data. In our system, the data is stored in a central blockchain that the Verifier has access to, and the user provides the Verifier only with decryption keys and indexes into 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 user's data; It can bound how deeply to investigate the graph of alibi's, if at all. We therefore satisfy this requirement by allowing the Verifier to choose how complex the verification process needs to be.

4.2 Attacks on the system

4.2.1 Identity replication

By repeatedly publishing fake transactions with the same ID, a malicious user can impede the Verification of certain transactions. This is because a Verifier will have to attempt to decrypt all transactions with the same ID until it finds one that decrypts successfully. A malicious user can therefore slow down the verification stage of certain transactions by flooding the blockchain with transactions with the same ID.

This is an abuse of the system than an attack. A malicious user will not be able to target specific users' transactions, because they are privacy protected. The malicious user can therefore gain no material advantage or compromise the security of the system by using an attack like this.

To avoid this attack, Miner nodes may reject any transactions with duplicate transaction IDs. Assuming sufficiently large transaction IDs, a duplicate ID is unlikely to happen through honest operation. Miners can assume that any transactions with duplicate IDs are malicious, and reject them.

4.2.2 Sybil attack

This system is vulnerable to a Sybil attack, where a single physical node may create multiple *pseudoidentities* [5]. A suitably powerful node, or motivated attacker, could create enough pseudonyms to create a subnetwork of nodes, each creating malicious transactions with each other. This would allow an attacker to falsify a believable location proof.

It has been proven that a central certificate authority is the only method of preventing a Sybil attack [5]. Other measures, such as web of trust, increase the difficulty involved in performing an attack, but the system remains vulnerable to a motivated attacker.

Bibliography

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

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

[5] Douceur, J.R., "The sybil attack", Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260, Springer, Heidelberg (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] Coulouris, George; Jean Dollimore; Tim Kindberg; Gordon 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