Decentralised location proof system

April 1, 2016

Abstract

1 Introduction

2 Previous work

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 [1, 2, 3].

Hardware techniques [1] operate by supplementing existing Wi-Fi access points with *femtocells* (small cellular antennae that connect to the 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 solution requires investment in additional hardware to supplement existing Wi-Fi access points. It also requires access to mobile providers user database to identify users locations.

Other proof systems [2] deploy software on Wi-Fi access points. Access points taking part in the location proof network become part of a 'group' with a shared group signature. Mobile nodes can request a location proof from an access point, who signs it with the group signature. Access points provide location proofs to nodes without ever learning the node's identity, thus protecting user's privacy. However, due to the systems reliance on the group signature structure, access points become a target for attack. Compromising an access point would allow an attacker to create false location proofs.

[2] considers a number of threats in their architecture:

- **Dishonest users.** A dishonest user tries to obtain location proofs that certify her presence at some place at a particular time even if she was not there. Dishonest users may achieve this goal by colluding with malicious intruders.
- **Malicious intruders.** A malicious intruder is not interested in obtaining location proofs for her own use but offers to help other users to get location proofs on their behalf in exchange for other benefits like money.
- **Curious APs and applications.** A curious AP tries to learn a users identity while the user is acquiring a location proof from the AP. Similarly, a curious application tries to learn more location information from a location proof than it really needs.
- **Malicious applications.** A malicious application obtains location proofs from its users and then tries to take advantage of these proofs to get unauthorised access to other applications.
- **Active and passive eavesdroppers.** An eavesdropper records and maybe modifies communication between users, proof- issuers, or applications.

Distributed P2P location proof systems also exist [3]. A central authority is used and acts as a certificate authority, issuing a unique ID to each person using the system. Unique identity is proved by users by providing SSN, drivers licence, or passport documents. This removes the threat of a Sybil attack [5], but makes the central authority a clear target for attack. The system uses three parties to provide location proofs; a mobile user who requests a proof, a fixed location authority, and a mobile user who acts as a witness. The system becomes vulnerable in the case where all three users collude, but is otherwise quite an interesting approach.

[3] also outlines a number of possible attacks on their system:

- **False presence:** A malicious user can create a fake location proof on his own, without being physically present at the location. The fake proof is supposed to resemble an actual proof, which the user could have actually collected from a valid location authority.
- **False timestamping (backdating, future dating):** In a backdating attack, the user and the location authority colludes to create a proof

for a past time. Conversely, in future dating, the location authority and a user colludes to generate a proof with a future timestamp.

- **Implication:** A location authority and/or a witnesses can falsely accuse a user of his presence at a certain location. In this case, the malicious location authority and witness colludes to generate a false proof of presence for the user.
- **False assertion:** A user can collude with a witness, and generate a falsely asserted location proof. The truth value in such a fake proof is reinstated with the assertion received from the other user.
- **Denial of presence:** A user can visit a location and at a later time, deny his presence at that location. In such a case, the user actually denies the validity of a certain location proof that has been been generated upon his presence at that particular location.
- **Proof switching:** The user is expected to have full access to all storage facilities on his mobile device. Hence, the user utilizes the legitimate proof and manipulates the information to create a false proof for a different location.
- **Relay attack:** A user can use a proxy to relay the requests and collect a location proof. Alternatively, a location authority can maliciously relay assertion requests with the witness not being present at the site.
- **Sybil attack:** A Sybil attack occurs when a single user generates multiple presence and identities [29]. A user can launch a Sybil attack by generating multiple identities representing a user and a witness and provide false endorsements for location proofs.
- **Denial of witnesses presence:** At the time of proof verification, the user can claim the absence of witnesses at the site or falsely claim an assertion to be counterfeit. The user and the location authority may also collude and claim the non-availability of witnesses.
- **Privacy violation:** An attacker may capture an asserted location proof generated for a user, and discover the identity of the user and/or the witness.

'OTIT' [4], a model for designing secure location provenance, can be used to compare existing location proof systems. This model defines the following requirements necessary for designing any secure location provenance scheme:

Chronological, Order Preserving, Verifiable, Tamper Evident, Privacy Preserved, Selective In-Sequence Privacy, Privacy Protected Chronology, and Convenience & Derivability.

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* [7].

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 1 provides an overview of the operation of the proposed system.

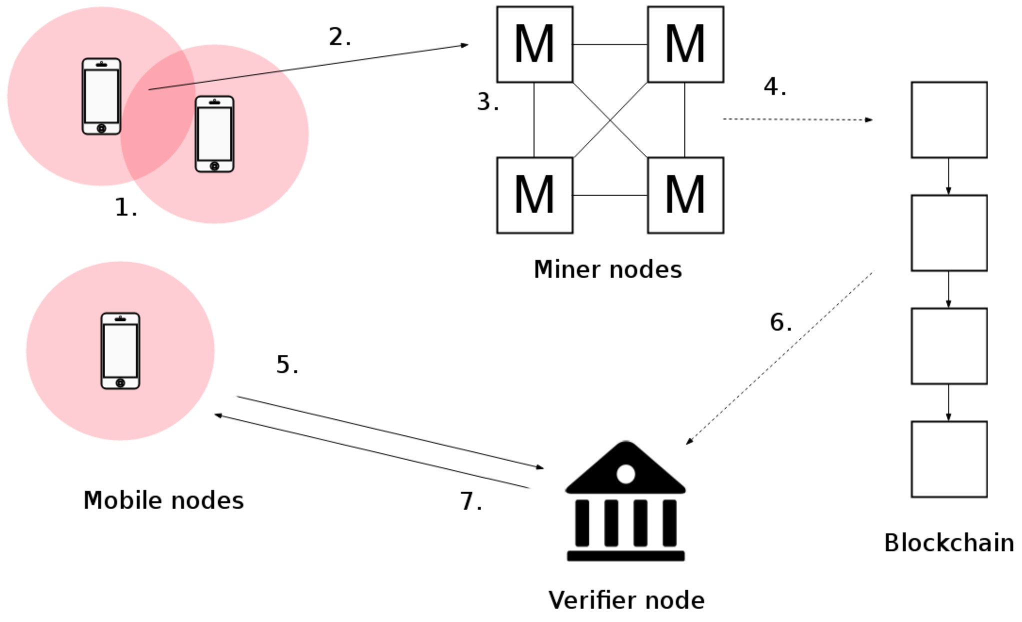


Figure 1: Design and operation overview

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 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

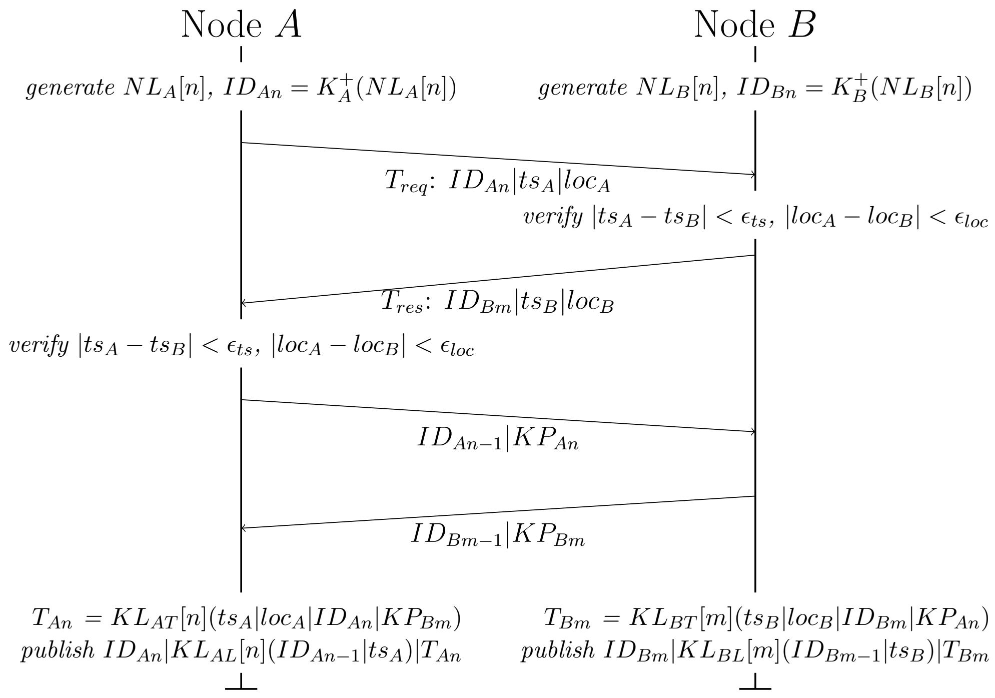
A *Nonce List* is simply a list of all of the nonces a mobile node has used so far to generate identities. When a mobile node generates a new identity, it must first generate a new nonce which is then encrypted to create the identity. The nonce is appended to the mobile node's nonce list, so it can be 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 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 $|ts_A - ts_B| < \epsilon_{ts}$, $|loc_A - loc_B| < \epsilon_{loc}$
$ID_{An-1}|KP_{An}$
$ID_{Bm-1}|KP_{Bm}$
$T_{An} = KL_{AT}[n](ts_A|loc_A|ID_{An}|KP_{Bm})$
publish $ID_{An}|KL_{AL}[n](ID_{An-1}|ts_A)|T_{An}$
$T_{Bm} = KL_{BT}[m](ts_B|loc_B|ID_{Bm}|KP_{An})$
publish $ID_{Bm}|KL_{BL}[m](ID_{Bm-1}|ts_B)|T_{Bm}$

Figure 2: Honest, successful transaction

Where:

$\mathbf{NL_A[n]}$ 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 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:

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

Where:

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.
$\mathbf{KP_{Bm}}$ 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:

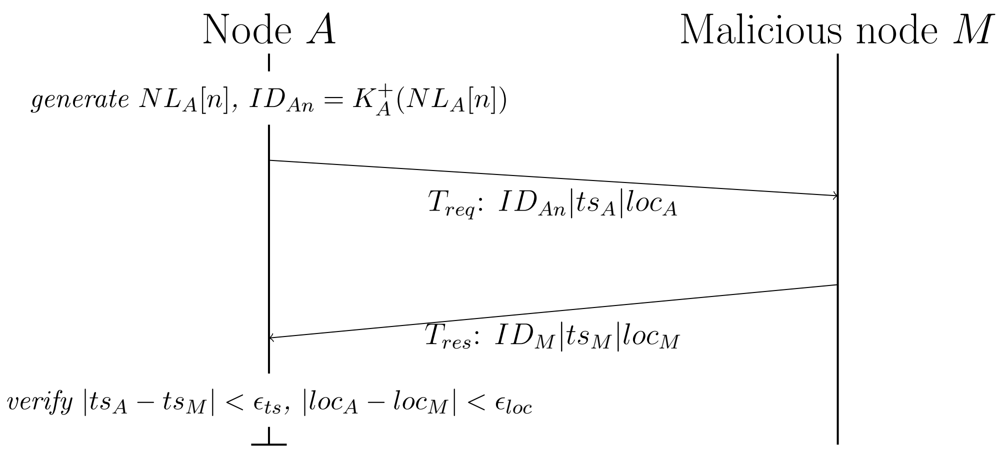
$$KP_{Bm} = (\{KL_{BL}[m], KL_{BT}[m]\}, KL_{BL}[m-1], \{KL_{BL}[m-2], KL_{BT}[m-2]\})$$

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, 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: 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 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.

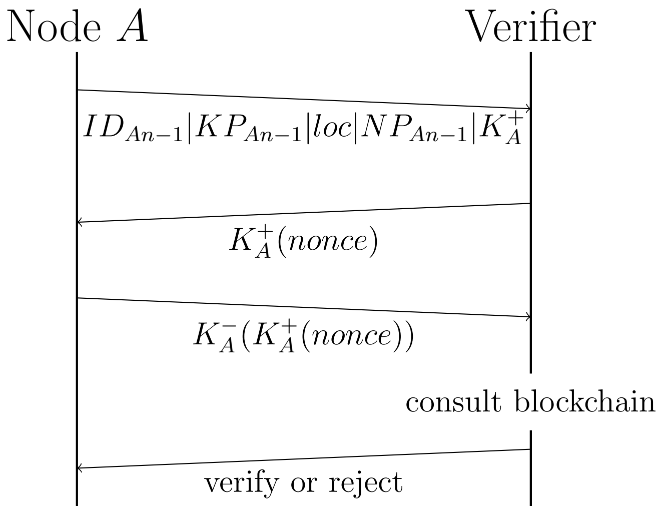


Figure 4: Verification request

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 nodes alibis have little or no alibis 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 alibis 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.

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 [7]. 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 [8]. 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.

5 Conclusions

References

[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. 712. 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. 251260, Springer, Heidelberg (2002)

[6] 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

[7] M. Swan, "Blockchain: Blueprint for a New Economy", Sebastopol, CA, OReilly Media, 2015

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