Decentralised location proof system

March 10, 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 WiFi 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 WiFi 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- ssuers, 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 witnesss 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, Vereifiable, 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 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 nodes anonymously agree upon their location and current time over a short-range network. Once both nodes agree upon these parameters, they each create an encrypted, privacy-protecting transaction logging their location and the alibi used to create the transaction. This transaction is published onto a public, append-only bulletin board known as a *blockchain*. Once given permission and decryption keys from a node, a *verifier* can determine whether or not the node is present at its claimed location.

Any node acting as an alibi in a transaction will share details of its most recent transactions with the other node in the transaction. This allows the verifier to be given permission to check the authenticity of all recent alibi's, and each of the alibi's recent alibi's, etc. 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.

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 malicious 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 node was the original creator of each identity. This is achieved using a public/private key pair for each node.

When a node is created, it generates a key pair. To maintain anonymoty, the node will use the public key to encrypt some nonces to create identities, and use these identify itself in a 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. The verifier calculates the node's identities using the public key and nonces, and retrieves the relevant transactions from the public blockchain. The verifier will also ensure that the node owns the private key associated with the provided public key, to prevent identity theft.

3.2 Transactions

During a transaction between two nodes, a number of different items must be shared and calculated. Figure 1 describes an honest, successful transaction, assuming an ad-hoc bluetooth network has already been set up between nodes A and B.

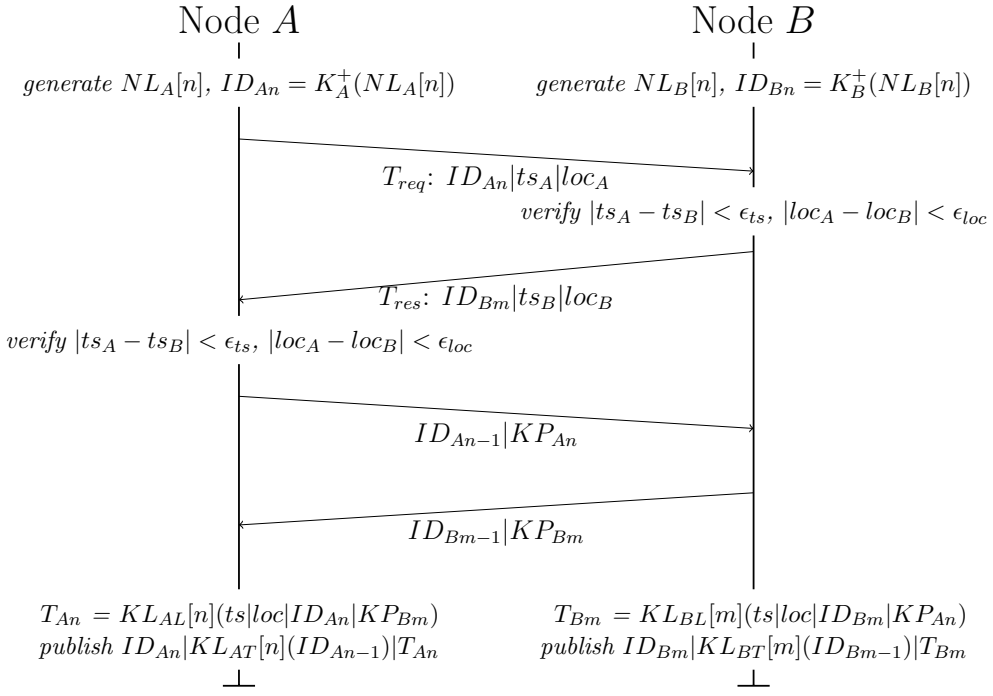

Figure 1: Honest, successful transaction

Where:

$\mathbf{NL_A[n]}$ is the nth nonce in A's *Nonce List*, which is used to generate ID_{An}.

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 at least initialised (if not already populated) before the transaction described in Figure 1 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).

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.

$\mathbf{KL_{AT}}$ is one of A's *Key Lists*, used to encrypt transactions.

$\mathbf{KP_{Bm}}$ 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 needs to publish it to the public blockchain. The data published must be identifiablein order to provide a verifier with some way of identifying specific transactions. Nodes will 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 node has not had its order altered. The transaction must therefore be accompanied by a link to the node's chronologically previous transaction. This link must be encrypted, otherwise a malicious user monitoring the public blockchain could monitor a user's transaction publishing activity. A's copy of the transaction log P_{An} will therefore be in the form:

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

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 B has used to encrypt his transactions; in order to preserve the "privacy preserved" 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 "provacy protected chronology" property, the transaction backwards-chaining must not be broken. This is why two separate *Key Lists* are maintained; KL_{A1} is used to encrypt the backwards-chaining link, while KL_{A2} 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 2, 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 $|ts_A - ts_M| < \epsilon_{ts}$, or $|loc_A - loc_M| < \epsilon_{loc}$.

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 does not reference the transaction of the alibi used to create it will be disregarded by any Verifiers.

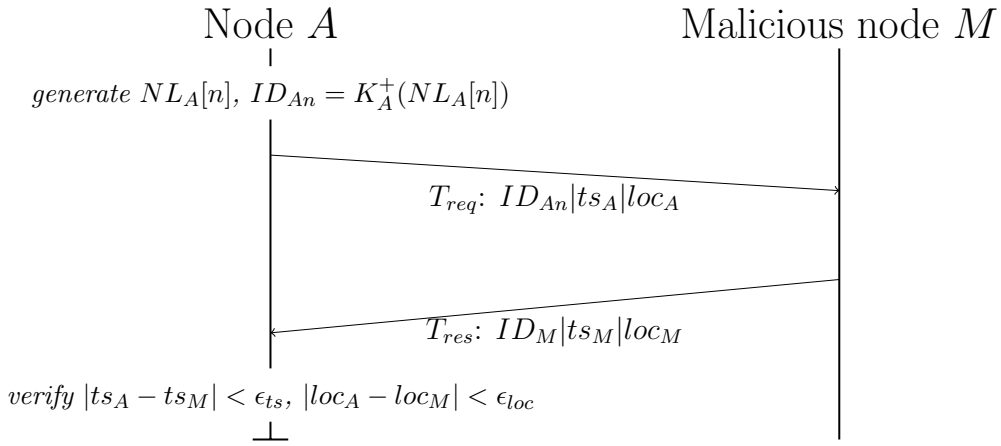

Figure 2: An aborted transaction due to malicious node M

3.3 Verification

In the verification stage, a node tries to prove its location to a Verifier 3rd party, e.g. a bank. To do this, the node sends the Verifier a number of parameters, as shown in Figure 3:

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

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

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

$\mathbf{NP_{An-1}}$ is A's *Nonce Packet*, a list of the nonces used to generate A's most recent IDs, from $n-1$.

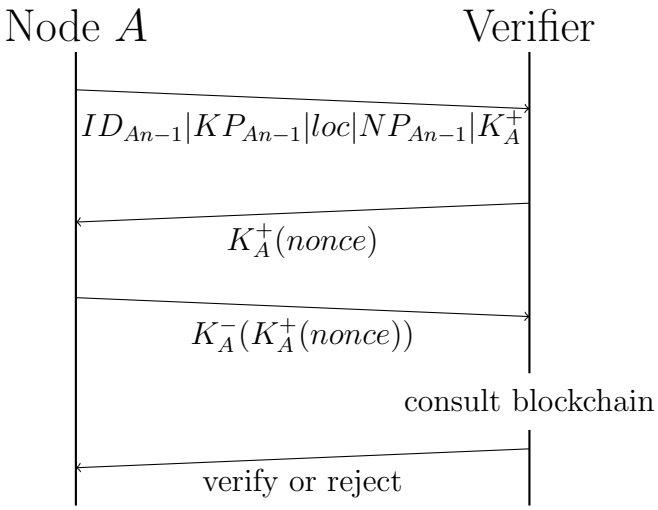

Figure 3: Verification request

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.

3.4 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.

4 Results

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