CS259 Project Report
Stanford University

Location Privacy via Private Proximity Testing

Mugdha Lakhani
Winter 2011

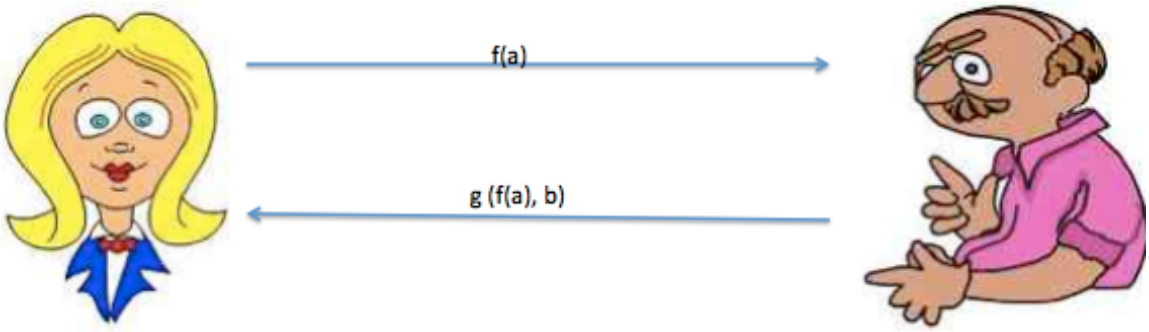
Abstract

The system I studied [1] presents three protocols for determining proximity of two nodes without compromising their location information. The authors present the idea of "location tags" generated from the physical environment to strengthen the privacy guarantees of the protocols.

Here I present the model of the first two of those protocols, and discuss possible attacks and fixes. Both the protocols are vulnerable to online dictionary attacks, identity spoofing and denial of service. I found that the use of location tags is necessary to provide very basic privacy guarantees. I suggest some fixes that make the protocols immune to these attacks.

Protocol 1: Passive Server

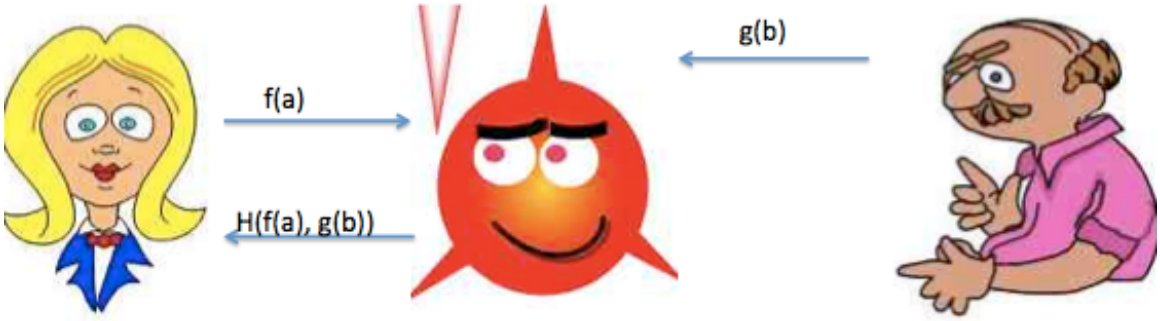
Both protocols rely on Facebook to provide public and private key pairs, and use ElGamal encryption. Here's what the protocol looks like after abstracting away the math. There are two agents Alice and Bob. a is Alice's location and b is Bob's. And functions f and g have the properties described below:



G(f(a),b) = 1 if a = b
= random otherwise
F(a) = random to Bob

Protocol 2: Active Server

Here, the server is an active participant. a and b hold same meanings as in protocol 1. Functions f, g, and H have properties as described below:



H(f(a), g(b) = 0 if a == b
= random otherwise

F(a) and g(b) are random to the server as long as no collusion with either Alice or Bob

Security Properties and Threat Model

I assumed that functions f, g and h have properties as described in the two pictures above, but also required, based on the protocol description, that the protocols have the following property, which I was interested in testing.

i) Privacy: At the end of several rounds of protocol 1 among any number of agents,(multiple simultaneous instances of the protocol, and each agent in multiple roles simultaneously), the responder (Bob) in each case, must not have any knowledge of a, and the initiator(Alice) must not have any knowledge of b if a != b. If a == b, then Alice should only know b upto a certain approximation (which depends on the granularity of the protocol).
ii) Confidentiality of messages
iii) Integrity of messages.
iv) Authentication: At the end of a round of protocol 1 or 2, if Alice thinks she was talking to Bob, then she actually was. (no identity spoofing).
v) Security: At the end of a round of protocol between Alice and Bob, Bob cannot cause Alice to compute the false answer from the protocol.

The threats were:

i) Online dictionary attacks: This is a big threat because the location space is small.
ii) Collusion: This not only causes leaking of unauthorized location information, but also makes dictionary attacks faster. Collusion is made easier because multiple instances of the protocol can run simultaneously, and each agent can be in multiple roles at once.

Murphi Model

I designed the murphi model to test privacy property, because that was the major contribution of the paper, leaving the other security properties for analytical analysis.

The model involves describing a probabilistic finite state system which was easier to model using Prism, but previous research [2] shows that FHP-Murphi can handle systems that are out of reach for Prism, especially those where state transitions involve arithmetic operations. The lengths of the Markov chains need to be bounded, however. I tried to model protocol 1 without using the FHP features of Murphi (see prot1.m), and found that the state space easily got too big for Murphi to handle. I then modeled protocol 1 and 2 using the advanced features of Murphi, notably the BPCTL and scalarset symmetry features. (see prot1_adv.m and prot2_adv.m). While the bpctl version of the model defines a finite state machine with probabilities assigned to every state transition, I was able to model the same behavior by storing the probability of being in the current state, for every agent, and updating it repeatedly.

Assumptions and Minutiae

1) Encryption and hashing are implemented, to preserve confidentiality and integrity of messages.
2) The public, private key pairs provided by Facebook at the time of initialization of the application are accurate.
3) There is one victim, (one server), and several attackers in the model. This is okay, since the behavior I was trying to capture doesn't change with this simplification.
4) Victim and server follow the protocol accurately; attackers are dishonest and use spoofed locations.
5) The attacker can be in the role of initiator or responder, but not in the role of server.
6) In either role, the attacker can collude with other responders and/or other colluders, which leads to four major types of collusions, out of which, I discarded the ones where an attacker (in either role) was colluding with attackers who are all in the role of a responder. These collusions were unfruitful because responders do not find out any information in these protocols, and depend in turn on the initiators to provide them some.

Modeling a dictionary attack

This was done by assigning a variable with each attacker, that kept count of how many locations he was yet to spoof, in an attempt to come across the right one. This number, say N, kept on decreasing as more and more rounds of the protocol were run, and in any round, the probability of guessing the right location was 1/N. This enabled me to find the probability of guessing the location, and when it would rise beyond a certain threshold, I would call it a successful dictionary attack.

Modeling collusion

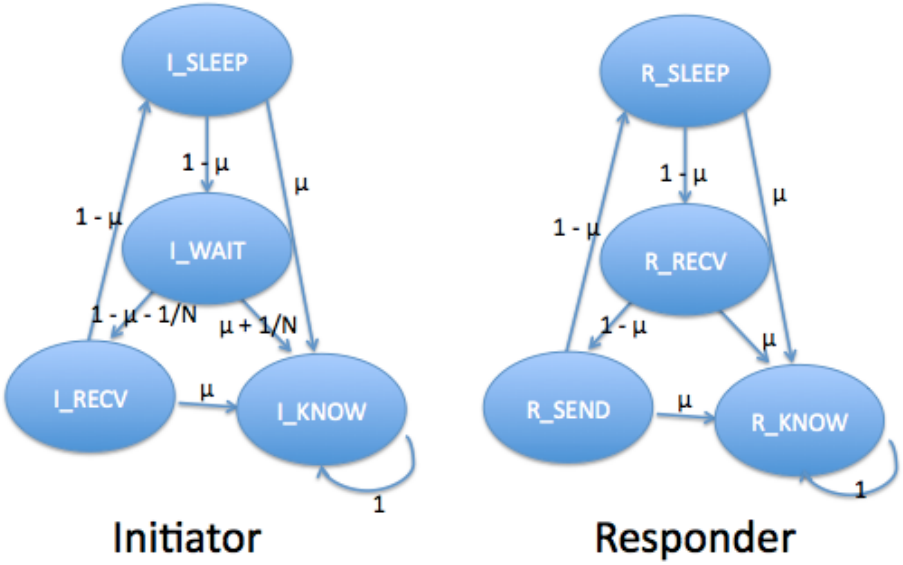
Collusion between attackers could be modeled in several ways (I used them interchangeably in my models):

i) Dividing up location space between attackers, or,

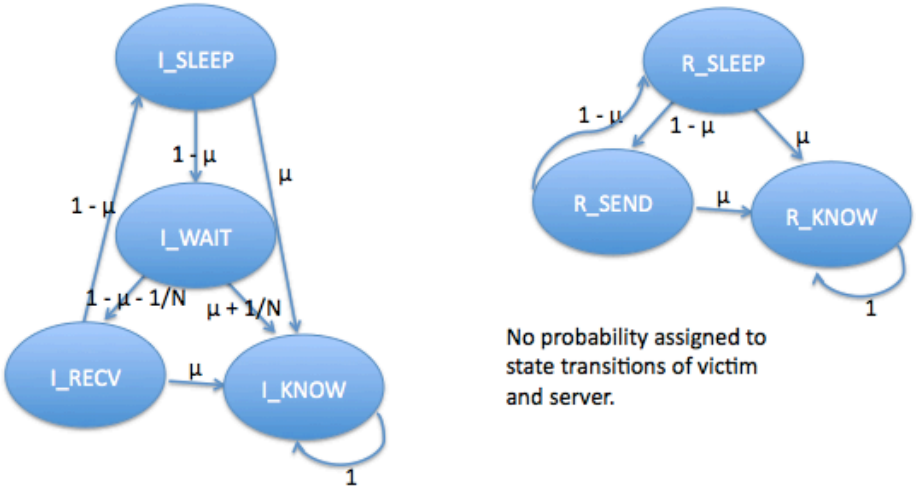
ii) Decrementing number of locations yet to explore by number of attackers, instead of 1 after each round of the protocol, or,

iii) Adding a state transition, with probability μ = (number of attackers – 1)/N, from every state to the error state(one in which location information is compromised), which signifies the event that some other attacker guessed the correct location enabling the attacker under consideration to go to the error state.

The state diagrams for entities in Protocol 1 are as follows:



And the one for the entities in Protocol 2 are as follows:



(Initiator states start with I_, responder states start with R_, and server states start with S_)

Note that the models are "attacker driven", that is, probabilities are only assigned to state transitions of whichever role the attacker is taking, and the victim and the server just follow through with the steps of the protocol. An attacker may take the role of an Initiator or a responder, but never takes the role of a server, because I am assuming that the server does not collude with either agent. This is because, protocol 2's privacy property fails under collusion with the server, and the authors were aware of the fact. Since it was not an intended use of the protocol, it wasn't worthwhile modeling that.

Invariants and Properties

The invariant was expressed as a PCTL formula of the form:

$Ps[\Phi \mathbf{U}{\leq}k\ \Psi]$ <= Threshold

Where Φ = *true,*and Ψ = *state where the victim's location is compromised,* and *Threshold* is the *maximum probability of reaching the error state.*

Analysis, Attacks and Fixes

Online Dictionary Attack:

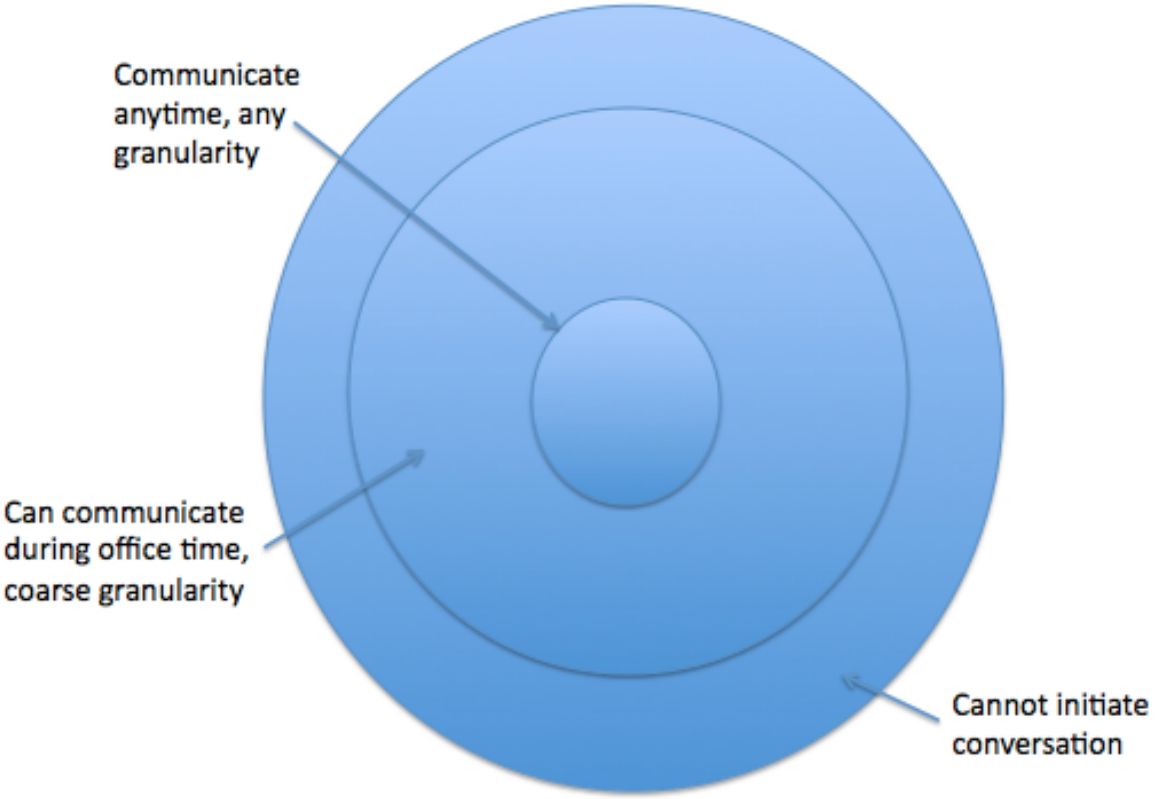
These were very easily found (with high probability), which was expected, given the small space of possible locations. They were found even more easily when the number of attackers was increased.

The best fix for this attack is to use location tags, because they prevent spoofing of locations altogether. (they make the probability of guessing the right location very small). However, if using location tags is not possible, then it is good to think about what granularity one is choosing for the protocols, because the finer the granularity, the larger the location space becomes, and the harder it gets to guess the right location each time. However, once a location has been guessed correctly, using a finer granularity means that the location has been guessed to a greater accuracy, thus causing a greater privacy breach. It is good to keep these things in mind.

Collusion

Collusion makes dictionary attacks faster, easier, and sometimes unnecessary. One cannot, unfortunately, completely prevent collusion without involving a trusted third party, which this protocol seeks to avoid.

Imposing restrictions on the time and extent of communication with certain "friends" can also reduce the efficacy of dictionary attacks. I propose diving the friend list into circles of trust, where increasing radii imply decreasing reliability on the people belonging to that set.



Keeping logs of previous activity can help one detect abnormal behavior, and stop online dictionary attacks while they are being mounted.

Identity Spoofing

This was found analytically.
In protocol 1, Bob sends his message encrypted with the Alice's public key. Everybody in Alice's friend list has her public key, and so anybody can pose as Bob by sending a location encrypted with Alice's public key, and cause the protocol to derive the wrong result.

In protocol 2, a shared key is used to encrypt messages between Alice and server, and Bob and server, respectively. If hashing is not implemented, then Carol can pose as Bob, use a random key to encrypt the message for the server, and cause the server to decrypt a wrong location, which in turn, causes the protocol to derive the wrong result. (The authors do not suggest or recommend hashing, but defending against this attack requires message authentication).

The fix here seems easy. The protocols can be changed to use a shared secret for encrypting messages, and exchanging nonces at the beginning of the conversation, similar to Needham Schroeder Lowe's protocol (except the key pairs are taken from Facebook), that are then to be included with every subsequent message, to ensure no identity spoofing is possible. However, the asymmetry of the protocol *depends on the fact that* no shared secret is used between the initiator and the responder (so that responder cannot decrypt and determine Alice's location). To fix this issue in

protocol 1, one can do the following: First exchange nonces similar to Needham Schroeder Lowe's protocol, and then, encrypt the nonces and the message of the original protocol (Alice's location encrypted with Alice's public key) with the shared secret. Why do we need a shared secret when we are already exchanging nonces? This is because, more often than not, in this protocol, the message received by Alice is random to her, and this can be achieved by encrypting any message with the wrong shared secret (so that after decryption with the right shared secret, it is a random number).

In protocol 2, the responder does not get any messages, so asymmetric encryption is not necessary, and two levels of encryption can be avoided. A shared secret is already used between initiator and server, and server and responder in protocol 2, so addition of nonces should prevent identity spoofing.

Denial of Service

The attacks described under Identity spoofing above, can lead to denial of service, if many attackers (in the role of responder) do the same, by sending spoofed locations, so that the initiator is unable to respond to legitimate requests. This is very much possible, given the fact that handheld devices have a limit on the number of devices they can communicate with, simultaneously.

Spoofing of locations can be prevented using location tags, again, and identity spoofing can be prevented using the fix described above.

Wrong Results

The attack described under identity spoofing can cause the protocols to derive wrong results. In Protocol 1, Bob can send a payload that causes Alice to wrongly derive that a == b. The authors do not admit this to be a flaw in the protocol as it does not cause a privacy breach. However, I feel that a protocol to determine proximity of locations should be able to say with confidence that proximity has been tested for, correctly. I could not come up with a fix for this problem that did not involve a trusted third party, however. Trusting third parties is getting riskier in today's world, and that was what the protocols were seeking to avoid. Implementing circles of trust, can however, reduce the impact of such attacks.

Security Properties Summary

Property	Protocol 1	Protocol 2
Confidentiality	Yes, assuming encryption correctly implemented	Yes, assuming encryption correctly implemented
Integrity	Yes, assuming hashing correctly implemented	Yes, assuming hashing correctly implemented
Authentication	No	No
Security	No	No
Privacy	Very weak under collusion Weak without collusion	None under collusion Weak without collusion

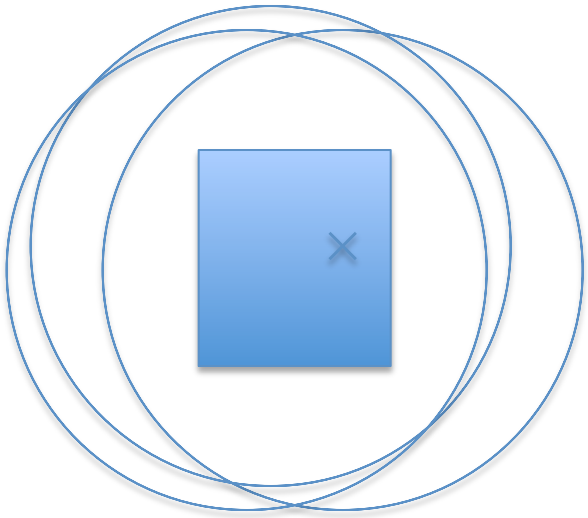
Side note: Triangulation Attack

Triangulation attack involves finding three locations **around** the victim, at some known distance, to determine the victim's exact location.

However, since both our protocols give location information approximated to the center of a grid square, one *cannot glean any more information from it* by finding more and more locations within the same grid square.

This is because if the grid is square, for example, with side l, then, for every location x that one finds out, the location of the victim, v, will be within a circle of $l*\sqrt{2}$. Now the region v lies in, will be the area of intersection of all these circles. But since all the circles have radius $l*\sqrt{2}$, the intersection will not be smaller than the square itself. So one cannot get closer than the grid square to the location in this way.

Here's a picture showing how the intersection of all circles is still bigger than the square:



Also, it is not possible to 'go around' the victim because the victim's location within the grid square is not known, nor is any direction or angle of v with respect x is known.

Thus, quantization of location space makes it impossible to mount a triangulation attack on the two protocols.

Conclusion

I found that use of location tags increases the strength of the protocols' privacy claim immensely. Without them, the protocols were found weak, and vulnerable to several attacks.

Avoiding a trusted third party has some unavoidable consequences, which can be made less unfavorable by selectively engaging in conversation with certain groups of people and varying granularity with time and place.
It was interesting to observe the tradeoff between usability (accuracy of the protocol) and privacy. It was also fun to learn probabilistic modeling on Murphi.

Limitations and Future Work

Given more time, I would have liked to model the fix suggested for preventing identity spoofing in the protocols. It would also be interesting to see how background information affects the efficacy of dictionary attacks. If some learning method could be developed for detecting dictionary attacks on the fly, it would be very useful in preventing information leaks. Also, finding a fix for preserving security of the protocol without involving a third party would be very welcome.

References

[1] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, DanBoneh. Location Privacy via Private Proximity Testing. In NDSS, 2011.

[2] Giuseppe Della Penna, Benedetto Intrigila, Igor Melatti, Enrico Tronci, and Marisa Venturini Zilli. Finite Horizon Analysis of Markov Chains with the Murphi Verifier.

[3] Giuseppe Della Penna, Benedetto Intrigila, Igor Melatti, Enrico Tronci, and Marisa Venturini Zilli. Bounded Probabilistic Model Checking with the Murphi Verifier.