Design and Evaluation of an Architecture for Location Privacy

Urs Hengartner
Cheriton School of Computer Science, University of Waterloo
uhengart@cs.uwaterloo.ca

July 24, 2008

Abstract

Many operators of cellphone networks now offer location-based services to their customers, whereby an operator often outsources service provisioning to a third-party provider. Since a person's location could reveal sensitive information about the person, the operator must ensure that the service provider processes location information about the operator's customers in a privacy-preserving way. We propose an architecture for location-based services based on Trusted Computing and Secure Logging. Trusted Computing lets an operator query the configuration of a location-based service. The operator will hand over location information to the service only if the service is configured such that the service provider cannot get access to location information using software-based attacks. This includes passive attacks, where the provider monitors the composition or the timing of information flowing into and out of its service, and active attacks, where the provider modifies or injects customer queries to the service. We introduce several requirements that must be satisfied by a location-based service to defend against passive attacks. Furthermore, we present Secure Logging, an auditing mechanism to defend against active attacks. We present an implementation of our architecture and demonstrate its usefulness by building two sample location-based services on top of it. The evaluation of our architecture shows that its processing delay is hardly noticeable and that timing attacks are very unlikely to succeed.

1 Introduction

The ubiquity of cellphones has led many operators of cellphone networks to offer location-based services to their customers, allowing these customers to map their current location, learn about interesting, nearby places, locate other cellphone users, etc. Often, an operator outsources the provisioning of a location-based service to a third-party provider. This outsourcing raises privacy concerns. Namely, the knowledge of a person's current or past locations could reveal sensitive information about the person's interests, her health, or her political preferences. Therefore, location information itself is sensitive, and the operator should have assurance that the service provider deals with location information about the operator's customers in a privacy-preserving way, based on a given privacy policy. So far, this assurance has relied on a legal contract between the operator and the provider. However, there has been no technical mechanism that lets an operator verify whether the provider adheres to the privacy policy.

In this paper, we present an architecture for location-based services that provides such a mechanism. Our architecture exploits Trusted Computing technologies [39] to let an operator gather information about the configuration of the platform that provides a location-based service. The operator hands over location information to the platform only if the platform is configured to implement an outlined privacy policy. Here, we are interested in the most constrained case where the policy states that the service platform must provide its service such that the service provider cannot learn any location information.

Unfortunately, giving a comprehensive guarantee that covers all possible attempts by a service provider to get access to location information is likely impossible. However, we can provide a weaker, but still useful

guarantee. Namely, we can guarantee that, using software-based attempts only, the service provider will not be able to learn fine-grained location information. This guarantee is superior to the non-technical guarantees offered by current location-based services.

The usage of Trusted Computing technologies to inspect the configuration of a service has already been suggested [26, 36]. We address two additional challenges. First, the earlier work has not considered how interactions of the service (e.g., a web server) with the backend infrastructure (e.g., a database) could lead to information leaks. We show that, in a passive attack, the service provider could learn location information by observing the composition or the timing of information flowing into and out of the service platform. Second, the earlier work includes an entire application, such as a web server consisting of thousands of lines of code, in the computing base that is remotely inspected and subsequently trusted. The size of such an application makes it likely that there is a vulnerability that a service provider could exploit. Therefore, we do not want to include an entire location-based service in the trusted computing base. Instead, only the part that needs direct access to location information (about 400 lines of code for our sample location-based services) should be included, but not, for example, the access-control part. However, this approach enables active attacks by a service provider, where the provider modifies or injects queries to the service to learn location information.

We make the following contributions:

- We build an architecture for the privacy-preserving processing of location information based on Trusted Computing technologies.
- We give a set of requirements that need to be satisfied to defend against passive attacks by a service provider.
- We introduce Secure Logging, an auditing mechanism that allows the retroactive detection of active attacks by a service provider.
- We present an implementation and evaluation of our architecture. The evaluation shows that the processing delay is hardly noticeable and that timing attacks are very unlikely to succeed.

This paper is an extended version of a paper that appeared in SecureComm 2008 [17]. The previous paper misses the implementation (Section 7), security analysis (Section 8), and evaluation of our architecture (Section 9). Furthermore, this paper presents a second sample location-based service built using our architecture (Section 6.2) and discusses design decisions in more detail. We presented a preliminary version of our architecture in a workshop paper [16], omitting the requirements needed to defend against passive attacks, the protocol to validate satisfaction of these requirements, and Secure Logging.

In the rest of this paper, we first present our system and threat models (Section 2). We then introduce our architecture (Section 3). Next, we examine two of its components in more detail, the Trusted Module (Section 4) and Secure Logging (Section 5). We then present two sample location-based services built using this architecture (Section 6). Next, we describe our implementation (Section 7), followed by a security analysis (Section 8) and an evaluation of our architecture (Section 9). Finally, we survey related work (Section 10).

2 System and Threat Models

In our system model, the operator of a cellphone network and the provider of a location-based service are separate entities. There are two infrastructures, the *cellphone infrastructure*, run by the operator of the cellphone network, and the *service infrastructure*, run by the provider of the location-based service. The cellphone infrastructure keeps track of the location of cellphones by observing which cell towers a cellphone

is connecting to or by directly getting location information from GPS-enhanced cellphones. When asked for by the service infrastructure, typically as a consequence of a query by a cellphone user, the cellphone infrastructure hands over location information, maybe in processed form, to the service infrastructure. Several network operators in the UK, such as Vodafone or Orange, provide their customers' location to service providers, such as mapAmobile [6] or World-Tracker.Com [41]. Sprint and Bell Canada use WaveMarket's Family Finder [40] to provide location-based services. Lots of existing location privacy research is also based on this system model [2, 7, 13, 24, 30, 32, 35].

There are two alternative system models. In the first model, a cellphone user installs an operator-independent client of a location-based application on her cellphone and has the cellphone interact directly with the service infrastructure. An example is Google Maps Mobile [12]. Our approach can also be applied to this kind of system model. The only change is that the cellphone would have to take care of the functionality that our approach assigns to the cellphone infrastructure. In the second model, cellphones interact with each other directly to implement a distributed location-based service in a privacy-preserving way or to achieve location privacy before accessing a centralized location-based service. Previous research [11, 15, 42], including research by the author of this paper [42], has focused on this model.

Our threat model consists of a service provider learning a user's fine-grained location. The provider can perform software-based attacks to extract this information from the service infrastructure, but no hardware-based attacks or attacks based on physical user observation. (We take a closer look at hardware-based attacks in Section 8.) These attacks are more expensive to perform. Also, defending against software-based attacks still gives us better privacy than what existing location-based services provide. We also assume that a service provider can observe the input and the output flowing into and out of the service platform and modify or inject queries sent to the platform. Finally, for efficiency reasons, we allow the service provider to learn a user's coarse-grained location. As it turns out, this happens only for some location-based services, not all of them.

Our threat model allows a service provider to learn the identity of its users. For some services, such as a service to locate interesting, nearby places, it is straightforward (ignoring billing challenges) to extend our approach such that the service provider does not learn identity information; the cellphone infrastructure simply anonymizes a query before sending it to the service infrastructure. For other services, such as a service to locate nearby friends, where the service infrastructure needs information about friend relationships, the solution is less obvious and topic of future research.

In addition to location and identity information, there are other types of information that might enable re-identification of a cellphone or of a wireless device in general, such as its MAC address [22], its transmission characteristics [8], or its preferred SSIDs [33]. In our system model, the cellphone infrastructure acts as a shield and makes sure that only location and identity information are forwarded to the service infrastructure.

In the rest of this paper, we assume that a user's location corresponds to her cellphone's location, and we use both variants interchangeably.

3 Architecture Overview

Figure 1 shows our architecture for location-based services. In the cellphone infrastructure, we leave away components not relevant in this paper, such as cell towers. The cellphone infrastructure consists of a set of modules and communication links between these modules, where both the modules and the links are under the control of the network operator. The same applies to the service infrastructure, which is under the control of the service provider. Communication between modules belonging to different infrastructures takes place over TLS with client and server authentication, which avoids sniffing, modification and injection attacks by outsiders.

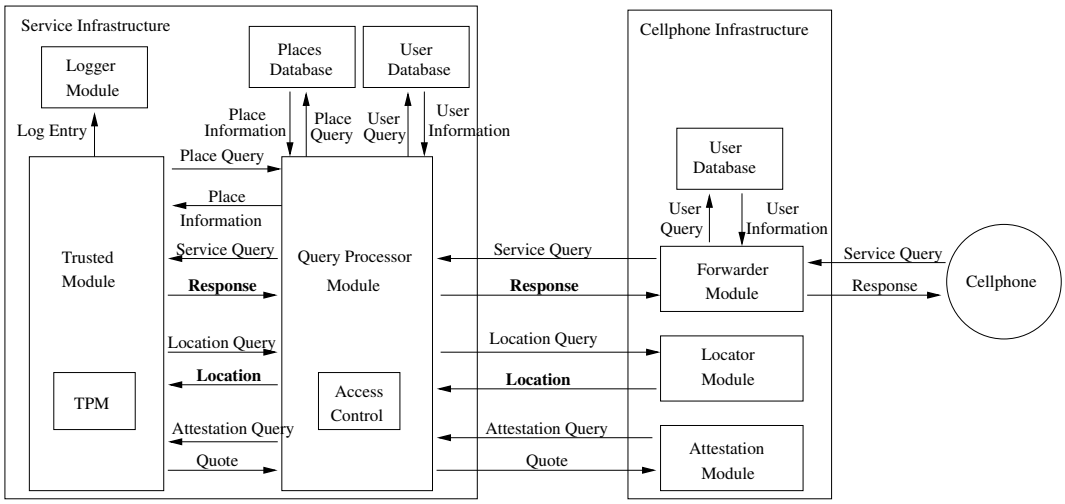

Figure 1: Architecture for location-based services. A cellphone sends a service query via the cellphone infrastructure to the service infrastructure, which uses the Trusted Module for processing the query and generating a response. Items in bold are encrypted (Response and Location).

3.1 Attestation Queries

Initially, the Attestation Module in the cellphone infrastructure queries the service infrastructure for the configuration of the Trusted Module (see Section 4.2 for details). It does so by sending an attestation query to the Query Processor Module in the service infrastructure, whose task is to receive different kinds of queries, to ensure that they are authorized, to process them if necessary, and to forward them to another module. In the attestation case, the module forwards the query to the Trusted Module and returns the response ("quote") to the Attestation Module. If this module is satisfied with the reported configuration (i.e., the configuration does not leak location information, see Section 4.1), it will give a symmetric key to the Trusted Module such that the Query Processor Module cannot learn the key. Later, the cellphone infrastructure will use the key for encrypting location information before handing the information over to the service infrastructure. This way, the Trusted Module, but not the Query Processor Module (or the service provider), can decrypt and learn location information.

3.2 Service and Location Queries

A cellphone user who wants to access a location-based service sends a service query to the cellphone infrastructure, namely, to the Forwarder Module. The Forwarder Module checks validity of the query based on information in the User Database (e.g., did the user sign up to the location-based service?). For accountability reasons, the module then signs the query and sends it to the location-based service, in particular, to the Query Processor Module. This module validates the signature of the query. For some services, such as a service that checks whether a friend is nearby, the module must also ensure that the cellphone user issuing the query is authorized to learn whether the queried person is nearby. Authorization information is stored in the User Database. If the query is authorized, the Query Processor Module sends a location query to the Locator Module in the cellphone infrastructure to retrieve the location information required for processing the query, such as the location of the user's and her friend's cellphone. The Locator Module returns location information only in encrypted form, namely, encrypted with the symmetric key introduced in Section 3.1. Next, the Query Processor Module hands over the location information, the operator's public

key, and logging information (see Section 5) to the Trusted Module.

The Trusted Module decrypts the location information. If necessary, as in the case of a query for interesting, nearby places or for traffic conditions, the module asks the Places Database via the Query Processor Module for location-specific information, such as road maps or conditions, weather information, shops, or restaurants. The Trusted Module then generates and signs its response and encrypts it with the operator's public key. To detect misbehaviour by the service provider, the module generates logging information, which it hands over to the Logger Module (see Section 5). The Trusted Module gives its response to the Query Processor Module, which forwards it to the Forwarder Module for decryption and signature checking. To thwart traffic-analysis attacks and to prevent unnecessary information from being sent to the cellphone, this module might have to filter the response (see Section 6). Finally, the Forwarder Module forwards the plaintext response to the cellphone.

3.3 Discussion

In our architecture, the Forwarder Module signs service queries and validates the signatures of responses. To achieve end-to-end security, a cellphone, instead of the Forwarder Module, could perform this functionality. Similarly, a cellphone could send attestation queries to the service infrastructure [29]. We do not adopt this option because of deployment and usability concerns. First, we would require a key distribution architecture and an infrastructure for disseminating information about approved platform configurations, which tend to be difficult to deploy. Second, users would have to install separate software on their cellphones, whereas our architecture allows a user to contact a location-based service simply by sending an SMS message to the cellphone infrastructure. Due to these concerns, we also have the operator send attestation queries to the service infrastructure, instead of a cellphone (see Section 4.2 for a more detailed discussion).

Apart from end-to-end security, is there an incentive for cellphone users to install software and to generate a cryptographic key pair on cellphones? As it turns out, this can increase users' privacy. In particular, the cellphone can encrypt a query so that its contents remain hidden from the operator (e.g., the operator will not learn what kind of interesting places the user is interested in). Similarly, the Trusted Module can encrypt its response with the cellphone's public key. However, hiding information from the operator is not part of our threat model. Therefore, in the rest of this paper, we assume that the cryptographic operations mentioned above are taken care of by the cellphone infrastructure and design our architecture accordingly. Along the way, we will point out what kind of changes are necessary if a cellphones took care of these operations.

The responses generated by the Trusted Module are of a textual nature. If required, a cellphone can process the response and generate a map, maybe in cooperation with the network operator or a third-party map service. If the Trusted Module generated maps, we would have to be careful to avoid information leaks (see Section 4.1).

4 Trusted Module

The Trusted Module is at the core of a location-based service and generates responses to service queries. The module is deployed by the service provider and has access to location information. Nonetheless, the service provider should not have access to this information. As stated in Section 2, we assume that the provider can perform only software-based attacks to gain access to the information. We discuss the set of requirements that need to be satisfied in order to defend against these attacks in Section 4.1. To ensure that a platform actually fulfills these requirements, a network operator uses the concept of remote attestation, which we discuss in Section 4.2.

4.1 Avoiding Information Leaks

To infer a cellphone's location, the service provider can mount various software-based attacks: First, the provider can observe the input given to the Trusted Module. Second, the provider can observe the output generated by the module. Third, the provider can observe the delay between input and output events. Fourth, the provider can examine the memory used by the module. We now examine each of these attacks.

4.1.1 Input Attacks

There are two kinds of input that are given to the Trusted Module and that could reveal a cellphone's location. First, there is actual location information. As discussed in Section 3.1, *we require that location information is encrypted before it is given to the Trusted Module*. The service provider does not know the decryption key, so the ciphertext is useless to the provider. Second, there is information about places (e.g., their locations), as retrieved from the Places Database. By examining which records the Trusted Module requests from the database, the provider could learn a user's location. We could avoid this attack by including the entire Places Database in the Trusted Module. However, this would drastically increase the size of the trusted computing base, whereas our goal is to keep it as small as possible. As discussed in Section 10, Private Information Retrieval (PIR) schemes also avoid this attack, but tend to be inefficient. We use a compromise that is more efficient, but that lets the service provider learn some information about a user's location. In particular, *we require the Trusted Module to cloak a user's location before accessing the Places Database.* For example, when the user is at the train station, the module determines a larger area that includes the train station, retrieves information about all the places in the cloaked area from the Places Database, and filters information about places that are too far away when generating a response. This way, the service provider can learn the area in which the person is, but not where exactly. Cloaking has been used extensively for providing location privacy [4, 7, 11, 13, 15, 30].

4.1.2 Output Attacks

There are two kinds of output from the Trusted Module that could reveal location information: Responses to service queries and log entries sent to the Logger Module. For the first kind, *we require that the Trusted Module encrypts responses with the operator's public key*. For the second kind, *we require that log entries do not contain location information.* Instead, they contain only information about service queries and about the public key used for encrypting responses to these queries (see Section 5).

Encrypting the response to a service query is not always sufficient to prevent the response from leaking location information. *We also require that there always is an output*, regardless of a cellphone's location. For example, in a parent-child tracking service, the parent gets alerted when the child leaves a boundary area. To implement this service, the Query Processor Module periodically invokes the Trusted Module, which determines whether the current location is outside of the boundary area, where the boundary area is known to the service provider. If the module returned a response only in the positive case, information would leak to the Query Processor Module. Therefore, the Trusted Module always has to generate a response and the content of the response must not leak any information. For example, the response could be the semantically secure encryption of the value zero or one.

Furthermore, *we require that the size of the output of the Trusted Module is determined by the user's cloaked location, not her precise location*. For example, when the Trusted Module uses cloaked location information to access the Places Database and filters the records received from the database, the response generated by the module must not allow the provider to learn how many of the records got filtered. See Section 6.2 for an implementation of this scheme.

Finally, the Trusted Module could output location information using some other means, such as writing the information to a file or to a display. *We require that the Trusted Module does not generate output*

information apart from the output shown in Figure 1.

4.1.3 Timing Attacks

The service provider could measure the delay between giving a service query to the Trusted Module and getting back a response. If different query outcomes resulted in different delays, measuring the delay could leak location information. *We require that the processing time of the Trusted Module is independent of the query outcome*. For example, for a service that checks for a nearby friend, the delay must be the same, regardless whether the friend is nearby or not. To strictly enforce this property, we would have to guarantee that the number of consumed CPU cycles is identical in either case, which is difficult to enforce in practice. Instead, in our sample location-based services, we ensure that the executed code path is identical in either case and that the only difference is the outcome of one or multiple statements comparing variables. For example, for a nearby-friends service, we have the module compute distance `dist` between two cellphones and then determine the query response as

```
response = (dist <= nearby);
```

where `nearby` indicates the distance considered nearby. In a more complex example, the following statements determine whether a cellphone at position `(x,y)` is within a rectangular area `(x1,y1)-(x2,y2)`:

```
cond1 = (x >= x1);
cond2 = (x <= x2);
cond3 = (y >= y1);
cond4 = (y <= y2);
sum = cond1 + cond2 + cond3 + cond4;
response = (sum == 4);
```

Our implementation (see Section 7) ensures that a service provider cannot perform a timing attack directly on the machine on which the Trusted Module is deployed, so the provider would have to perform a remote timing attack. Previous research [3] has shown that such attacks can be successfully mounted. However, the previous research relies on the existence of (remotely observable) timing differences of thousands or millions of CPU cycles, depending on the expected error. In Section 9, we show that the observed timing differences for the code examples shown above are orders of magnitudes smaller than what is required for a successful timing attack.

4.1.4 Memory Attacks

The service provider might try to extract location information or the symmetric decryption key from the memory used by the Trusted Module. *We require that the provider cannot access this memory using software-based mechanisms.* In our sample location-based services, we deploy the Trusted Module on a dedicated machine and configure the machine such that the provider cannot log in to it (see Section 7).

4.2 Remote Attestation

In Section 4.1, we came up with a set of requirements that the Trusted Module must satisfy to prevent location information from leaking. Of course, the developer of the Trusted Module, who could be identical with the service provider, might not actually address these requirements in the implementation. To ensure that our requirements are being satisfied, we use remote attestation.

Remote attestation is a feature proposed by the [39] (TCG). With the help of an inexpensive chip (in TCG terminology, the Trusted Platform Module (TPM)) soldered to the motherboard of a computer, this feature allows an entity to remotely gather information about the code running on this computer. Starting at boot

AM → TM:	*nonce*
TM:	generate key pair ($Public_{Transfer}$, $Private_{Transfer}$), measure $Public_{Transfer}$ in TPM, hand over *nonce* to TPM, retrieve quote
TM → AM:	measurement list ML, $S_{Private_{AIK}}(nonce||\text{digest}(ML))$, $Public_{Transfer}$
AM:	check quote signature, re-compute and approve ML, abort if failure
AM → TM:	$E_{Public_{Transfer}}(SymmetricKey)$
TM:	retrieve $SymmetricKey$, erase $Private_{Transfer}$

Table 1: Transfer protocol for symmetric key. AM denotes the Attestation Module and TM the Trusted Module. $S_{Foo}()$ is a signing operation with key Foo, $E_{Bar}()$ an encryption operation with key Bar.

time, each piece of running code (including BIOS code) gathers information about the next piece of code to be loaded and executed and stores this information in the TPM. This process is called a measurement. The TPM guarantees that measurements cannot be reverted (using software-based mechanisms). Therefore, we end up with a chain of measurements, rooted at the static root of trust, which typically is part of the computer's BIOS. If the entity asking a remote computer to perform a remote attestation trusts this root, the TPM, and each piece of loaded code, the entity can retrieve the measurement list from the TPM and infer what code is loaded and executed on the computer. In practice, a TPM does not store a measurement list, only a cryptographic digest, and the actual list is kept by the measuring code (see Sailer et al. [36] for details). A remote attestation returns the list and its digest, signed by the TPM, to the querying entity. A signed digest is called a quote.

Going back to the Trusted Module, the network operator exploits remote attestation to retrieve the measurement list from the module, which will allow the operator to learn the configuration of the module. The operator compares the received list to a list of approved measurements. Software is in an approved state if it is in a given state (e.g., a particular binary with a set of configuration parameters), as in the case of a kernel, or if it satisfies the requirements discussed in Section 4.1, as in the case of a component implementing a particular location-based service. To build the list of approved measurements, the operator must have had access to the software (including source code, binaries, and compiler, see Section 7 for details) beforehand for inspection purposes. If the operator can validate the received list of measurements, the operator will hand over the symmetric key that allows decryption of location information to the Trusted Module.

We show the protocol for this transfer in Table 1; it is partially based on earlier work [28, 36]. We assume that the configuration of the Trusted Module has been measured in the TPM as part of the boot process. When asking for a remote attestation, the Attestation Module generates a nonce and sends it to the Trusted Module, which generates an asymmetric key pair. The public key, $Public_{Transfer}$, will be given to the Attestation Module, which will employ it to transfer the symmetric key used for encrypting location information to the Trusted Module. By measuring $Public_{Transfer}$ in the TPM, the Trusted Module guarantees to the Attestation Module that the key pair was created by the Trusted Module and that the private key, $Private_{Transfer}$, is kept within the module and will be erased from memory after the transfer, as a defense against cold-boot attacks [14]. (We do not use the TPM's sealing option [39] for storing $Private_{Transfer}$ across attestation requests, since this option only reduces, but does not completely eliminate the window of vulnerability.)

The Trusted Module hands over the nonce received from the Attestation Module to the TPM and requests a quote. The returned quote includes the nonce and is signed with a private key embedded in the TPM (in TCG terminology, the private key of the Attestation Identity Key (AIK) pair). The Attestation Module knows the corresponding public key. The Trusted Module returns the quote, the measurement list, and $Public_{Transfer}$ to the Attestation Module, which recomputes the measurement list based on the expected

configuration of the Trusted Module. The Attestation Module cannot simply look at the quoted digest, since this value depends on the order in which the OS running on the Trusted Module's computer loads kernel modules and applications, which can differ slightly across reboots. If there are any values in the measurement list that are unknown to the Attestation Module, the module aborts the protocol. Else, the Attestation Module uses $Public_{Transfer}$ to encrypt the symmetric key and sends the ciphertext to the Trusted Module. In addition to $Public_{Transfer}$, the Trusted Module also transfers the public key for the signing key that it will use for signing responses to service queries (not shown).

The described protocol is executed whenever the platform of the Trusted Module is rebooted. An alternative is to execute the protocol only once and to use the TPM's sealing option for storing the symmetric key across reboots. Sealing allows encryption of the symmetric key such that it can be decrypted only when the Trusted Module is in the same configuration as at sealing time. This makes it possible to store the sealed key in permanent storage. However, as mentioned above, the configuration of the Trusted Module could change after a reboot, which means that the unsealing operation would fail.

Another alternative is to have a cellphone ask the Trusted Module for a quote and validate the returned measurement list, after which the cellphone would provide a (user-specific) decryption key for location information to the Trusted Module. Previous research [9, 29] has demonstrated the feasibility of having a cellphone validate measurement lists. However, the approach raises several deployment and usability challenges. First, users would have to install software on their cellphones, which they do not have to in our approach. Second, we cannot expect users to come up with an expected configuration of the Trusted Module, so we would need a trusted third party. This includes a mechanism for the third party to notify cellphones of changes in the configuration, as in the case of software updates. Third, the network operator already has access to location information and likely has a contract with the service provider that lets the provider offer location-based services on behalf of the network operator. In this setup, having the operator perform remote attestation seems the logical approach, instead of getting users involved and potentially confused. Fourth, if a cellphone cannot validate the measurement list, automatically denying the user from accessing the service could be wrong, as in the case of an emergency. Designing an appropriate user dialog is hard, since users tend to click away messages pointing out potential security problems, as experienced for expired or mismatched SSL certificates.

An operator might repeat remote attestation periodically (e.g., in an attempt to detect time-of-check, time-of-use (TOCTOU) attacks). Here, the Trusted Module also has to hand over previously used $Public_{Transfer}$ so that the Attestation Module can recompute the measurement list.

5 Secure Logging

The Trusted Module invokes the Logger Module to keep a log of its actions. These actions are requested by the Query Processor Module. With the help of the log, we can detect wrongly requested actions, that is, misbehaviour by the Query Processor Module (i.e., the service provider). For example, this module is expected to run access control for some location-based services, such as ensuring that only authorized people can track a person. If the module did not run access control and allowed, for example, the service provider to track the person, a corresponding log entry would make the person aware of this violation, and she would alert the network operator.

An alternative approach is to run access control in the Trusted Module. However, we would like to keep this module as simple as possible, which decreases the likelihood of a vulnerability in the module and also makes it easier for an operator to check the module's correctness. Furthermore, a service provider could be reluctant to implement the entire service within the Trusted Module, since part of this implementation might be proprietary, and the provider does not want to reveal it to an operator for inspection.

Our goal is to log each access to a person's location such that log entries cannot be tampered with or

Counter		Log entry			
TPM	Software				
0	0	AIK-certified signature verification key			signature
0	1	user id	query digest	public key digest	signature
0	2	user id	query digest	public key digest	signature
0	3	special log entry			signature
1	0	AIK-certified signature verification key			signature
1	1	user id	query digest	public key digest	signature
1	2	special log entry			signature
2	0	AIK-certified signature verification key			signature
2	1	user id	query digest	public key digest	signature

Table 2: Log format. A signature covers the counter and the log entry.

removed. For each access to a person's location, logging should enable this person to learn 1) the service query that triggered this access and 2) the public key that was used for encrypting the response to this query. In Sections 5.1 and 5.2, we discuss how we support these requirements. In Section 5.3, we show how we use the logged information to detect misbehaviour.

5.1 Logged Information

A log entry is generated by the Trusted Module whenever it accesses somebody's location. An entry contains an identifier unique for the user whose location was accessed, a cryptographic digest of the service query that triggered this access, and a cryptographic digest of the public key used for encrypting the module's response (see Table 2). It is possible to log additional information, such as the accessed location information (in encrypted form) or the response to a query, but this is not required by our threat model.

The user identifier will be used during log checking to learn whose location information was accessed. This identifier is communicated to the Trusted Module by the Locator Module, together with the (encrypted) location information. A message authentication code protects their integrity.

The query digest is computed by the Query Processor Module and given to the Trusted Module. The Query Processor Module needs to store the corresponding service query.

The Query Processor Module also hands over the operator's public key to the Trusted Module so that the module can encrypt the response to a query. A malicious Query Processor Module could hand over its own public key, which would allow it to decrypt the response. Therefore, the Trusted Module, not the Query Processor Module, computes the digest of the received public key to be included in the log entry. We let the Query Processor Module compute the digest of a (signed) query because the module gains no advantage from cheating here.

A single query can result in multiple log entries. For example, in a nearby-friends service, if Alice tries to locate Bob, the Trusted Module logs accesses to both Alice and Bob's location, each entry containing a digest of Alice's query. The operator inserts a nonce into a query to detect replay attacks by the Query Processor Module (e.g., in a timing attack).

5.2 Tampering and Removal Attacks

To protect log entries against tampering, the Trusted Module signs each entry before handing it over to the Logger Module. The Trusted Module logs the public key for the signing key in certified form whenever the module is initialized. The certificate is signed with the TPM's AIK private key. We cannot use this

key directly for signing a log entry, since it can be used only for signing keys. Also, private-key operations executed by the TPM are expensive (844 ms for a signature), so we avoid them during the processing of a service query.

To protect log entries against their removal, the Trusted Module includes the current value of a monotonically increasing counter in each entry and increases this counter after generating an entry. Here, the module needs to be able to reliably determine the value of this counter in case of a restart of the module. A possible approach is to use a TPM v1.2 monotonic counter, which is a counter within the TPM that can never be decreased. A problem with such a counter is that a TPM might support increases only once every five seconds to avoid burnouts. To increase logging frequency, we divide time into epochs. An epoch corresponds to the time between the initialization of the Trusted Module and its shut down. We use a TPM v1.2 monotonic counter to denote the current epoch and a software-based monotonic counter to denote time within an epoch. The Trusted Module includes the current values of both counters in a log entry. The module increases the TPM v1.2 monotonic counter only during module initialization. At the same time, the software-based counter is set to zero. This counter is increased after generating a log entry, which results in a global order among the log entries.

The above algorithm ensures that, given a log entry, we can detect removal of earlier log entries that have the same TPM v1.2 monotonic counter value. However, the algorithm does not protect against the removal of more recent log entries (i.e., with larger software-based or TPM counter values) or against the removal of log entries with lower TPM v1.2 monotonic counter values. For example, given a log entry with a TPM v1.2 monotonic counter value of 5, all we can say is there are must be log entries with TPM v1.2 monotonic counter values of 0, 1,..., 4, but not how many of each kind. There could also be log entries with TPM v1.2 monotonic counter values of 6, 7,... Let us discuss how we address these two attacks.

For the first attack, we want to ensure to a cellphone user that the last log entry seen by her when checking the log is the most recent log entry generated by the Trusted Module for an access to her location information (or close to the most recent one, since new entries can be added while the user is checking log entries). We can satisfy this condition by making the user send a query that accesses her location information to the location-based service before the user checks her log entries. This way, the Trusted Module will create a new log entry, and the user can verify whether there is an entry for her query in the log. If so, the log is up to date.

The second attack can be avoided by having the Trusted Module generate a special log entry whenever the service provider makes the module shut down. This way, an increase in the value of the TPM v1.2 monotonic counter, as observed in the log, is always preceded by such a special entry, and missing log entries can be detected (see Table 2). The absence of a special entry does not necessarily imply malicious behaviour. For instance, the Trusted Module could have crashed or there might have been a power outage, which prevented the module from creating the special entry. However, an operator should get suspicious if there are many missing special log entries. Then, either the location-based service is unreliable or there is widespread removal of log entries. Neither of the options is in the operator's interest, and the operator should stop using the service.

5.3 Log Validation

Consistent with the architecture described in Section 3, we assume that the operator will take care of the cryptographic operations required for log validation. If a log entry is considered valid, the operator will forward the query that triggered access to a particular user's location to this user for inspection. In more detail, log validation consists of the following steps:

Having a user review log entries in the context of location-based services has been suggested in previous work (e.g., by Prabaker et al. [34]). There, the assumption is that a location-based service is trustworthy, and the goal of reviewing is to allow the user to validate her access policy. Here, reviewing is done for validating

both the access policy and conformance of the service provider.

1. The user issues a service query, where processing of the query requires access to her location information. This will create a log entry.
2. The operator downloads the log (i.e., the entries added since the previous validation) from the service platform. The operator checks the digital signature of each entry and ensures that the counter values are monotonically increasing and complete.
3. The operator determines the log entries relevant to the user based on the user identifier and asks the service provider for the corresponding service queries. The operator ensures that the digests of the received queries match the logged digests and that the queries are properly formed and fresh. The operator also checks whether the query issued in step 1) is in the log.
4. The operator ensures that the logged digests of the public key match the digest of the public key of the operator. If so, the query is forwarded to the user.
5. The user inspects the received queries. If she detects an unauthorized query, she will alert the operator.

The above protocol differs slightly in case a user's cellphone is set up to encrypt service queries so that their contents remain hidden from the network operator. Here, upon log checking, the Query Processor Module encrypts a query with the cellphone's public key before giving it to the operator. The operator forwards the encrypted query and the log entry to the cellphone, which decrypts the query and ensures that its digest matches the digest in the log entry and that the public key used for encrypting the response corresponds to the cellphone's public key.

6 Sample Location-Based Services

Let us now discuss how we build two sample location-based services in our architecture.

6.1 Nearby Friends

This service allows a user to learn whether a friend of hers is nearby. The user sends a query listing the name of the friend to the Forwarder Module, which signs the query, adds the (encrypted) location of the user's cellphone, and forwards it to the Query Processor Module. (Including the location with a query is an optimization to the architecture discussed in Section 3.2.) The Query Processor Module retrieves the friend's policy from the User Database and ensures that the friend has granted access to the user. If so, the module requests the location of the friend's cellphone from the Locator Module. Next, it hands the two locations and the operator's public key over to the Trusted Module.

The Trusted Module uses the symmetric key received during attestation to decrypt the two locations and determines whether they are nearby. Depending on the result, it signs a byte of value either zero or one with its private key, encrypts everything with the operator's public key, and returns the ciphertext to the Query Processor Module, which sends it to the Forwarder Module. This module decrypts the ciphertext and sends the result to the cellphone. The Trusted Module also logs two accesses to the user's and her friend's location.

6.2 Interesting Places

This service returns interesting places (e.g., restaurants) close to the user's current location. Let us assume that the owner of the cellphone is willing to install separate software on her cellphone, which signs and encrypts queries and checks and decrypts responses to achieve end-to-end security. This approach also

results in additional user privacy, since it shields information from the network operator. (However, this is not required by our threat model.)

The cellphone sends a signed and encrypted query indicating what kind of places the user is interested in to the Forwarder Module. The query is encrypted so that the network operator cannot learn the user's interests. The module adds the cellphone's location and forwards the query to the Query Processor Module. This module decrypts the query and hands over the type of places, the cellphone's location, and the cellphone's public key, stored in the User Database, to the Trusted Module. The module decrypts and cloaks the location and retrieves the locations of a set of relevant places within the cloaked area from the Places Database. Next, the Trusted Module allocates a chunk of memory, one bit for each received place. The module looks at each place, decides whether it is close enough to the cellphone's actual location, and sets the corresponding bit in the chunk to either zero or one. Then it encrypts the entire chunk of memory with the operator's public key, and it encrypts the location of each place separately with the cellphone's public key. All the ciphertexts are sent to the Query Processor Module and forwarded to the Forwarder Module. This approach guarantees that the Trusted Module executes the same code path and generates the same amount of output, regardless of which and how many places are relevant.

The Forwarder Module filters the responses. It decrypts the chunk of memory indicating which of the (encrypted) places are relevant and forwards only the relevant ones to the cellphone, which decrypts them. Again, by encrypting the places, we prevent the network operator from learning the user's interests. The filtering done by the Forwarder Module ensures that no unnecessary traffic is sent to the cellphone, for which the user might get charged.

7 Implementation

We implemented our architecture and the nearby-friends and the interesting-places services in C on Linux. We use 2048 bit RSA keys for public-key cryptography, AES in CBC mode with 256 bit keys for symmetric-key cryptography, and HMAC for message authentication codes. The non-TPM-based cryptographic operations exploit OpenSSL.

Our implementation of the Trusted Module consists of three parts. On the hardware side, there is a Lenovo T60 ThinkPad with an Intel Core Duo processor (1.83 GHz), which comes with an Atmel TPM v1.2. On the software side, the OS is Ubuntu Linux 2.6.20.3, extended with IBM Research's Integrity Measurement Architecture (IMA) patch [20]. This patch provides support for remote attestation by enabling the kernel to measure kernel modules and applications, as well as applications to measure files. Furthermore, we built two application-layer components that interact with the Query Processor and the Logger Modules. These "TM components", one for each location-based service, use TrouSerS 0.3.0 [19] for application-layer communication with the TPM. Each component includes about 600 lines of attestation code. The component for the nearby-friends service consists of an additional 350 lines of code that process location information (420 lines of code in case of the interesting-places service).

As mentioned in Section 4.1, we have to avoid that the Trusted Module leaks location information to the service provider. To prevent leaks to other applications running on the same computer, maybe via side channels, the Trusted Module is the only application that runs on the ThinkPad. Any other applications should not be approved during remote attestation. In the future, more advanced features that are becoming available in x86 processors will make it easier to ensure that applications running on the same machine as the Trusted Module will not interfere with the module [28].

An advantage of having a separate machine for the Trusted Module is that we do not have to worry about other users or processes that (maybe legitimately) want to use this machine. Instead, we can simply lock them out, and we do not allow any user (including superuser) logins. This way, we do not have to worry about users gaining superuser rights, users examining the running TM components, or limiting the

superuser's capabilities, such as preventing the superuser from examining the memory of the TM component. Furthermore, if users are not able to log in, timing and TOCTOU attacks (see Section 8) become more difficult.

There are multiple ways to forbid logins. An option that requires only a small change (which is important, see below) is to disable them in /etc/shadow. In order to manage the Trusted Module, we boot an OS that resides in a different disk partition. To detect attacks on the /etc/shadow, the TM component measures this file. Assume a malicious provider changes the file and boots the platform. If an operator asks for a remote attestation, /etc/shadow will not look to the operator as expected. The superuser could try to reverse the changes to /etc/shadow by trying to log in to the machine before the TM component is started. However, the login (and the fixes to /etc/shadow) will invoke applications that are not approved by the operator (e.g., /bin/login). If the superuser changes the init application (which is started by the kernel) such that it does not run these applications, these changes will also show up in the measurement list. The same is going to happen when the TM component is modified to measure a fake shadow file.

As an additional precaution, we run only the minimal number of processes, in addition to the TM component, on the machine (e.g., no SSH daemon). An operator should abort remote attestation if additional processes show up in the measurement list. In total, there are about 50 processes and threads, most of them kernel level. We use Ubuntu's default kernel, configuration, and development tools for compiling the kernel and the TM components. It is possible to trim down the kernel configuration so that the number of processes and threads gets even smaller. However, this trimming down can be dangerous. In particular, when it is done by the service provider, the provider might try to sneak information leaks into the kernel that are difficult to detect by the network operator. Similarly, when using a non-standard compiler for compiling a TM component, the service provider could compromise the binary. By using software that is not under the control of the provider, we avoid these attacks and make it easier for the network operator to come up with a list of approved software. Similarly, keeping the amount of software that is run on the machine small simplifies maintenance of this list. Furthermore, with the help of property-based attestation [25], an operator can delegate maintenance to a trusted third party.

Finally, we turn off suspend and disable swapping to ensure that memory never gets written to disk. Our machine has ample memory (2 GB) and no need for swapping.

8 Security Analysis

Our implementation of the Trusted Module is susceptible to time-of-check, time-of-use (TOCTOU), hardware-based attacks, and cold-boot attacks.

In a TOCTOU attack, the attacker manages to change the configuration of the Trusted Module after remote attestation has taken place. This requires the attacker to have access to the Trusted Module and the presence of an exploitable vulnerability, such as a buffer overflow. In our implementation, we make this attack hard by keeping the TM component small and by giving the service provider neither local nor remote login access to the platform of the Trusted Module.

TPMs are not designed to withstand hardware-based attacks. Whereas hardware-based attacks are outside our threat model, recent work [23] has demonstrated their feasibility. There are three ways to address these attacks: First, the TCG defines a special kind of credential, the Platform Credential, that contains information about the hardware configuration of a platform. This credential can be retrieved during remote attestation, so a network operator can abort if the service provider uses a vulnerable platform. Second, recent extensions to x86 processors have made it possible to get rid of the static root of trust and to use a dynamic root of trust, which makes hardware-based attacks more difficult [23]. However, this approach might make timing attacks easier. Moreover, it would have high performance overhead. For each query, the Trusted Module would have to perform an unsealing operation, which takes hundreds of milliseconds with current

TPMs [27], to get access to the symmetric key. Third, we can deploy the Trusted Module in a tamperproof coprocessor, such as the IBM 4764 [18]. However, these coprocessors tend to be more expensive and slower than current server hardware and can have usability problems [26]. Even if we used such a coprocessor, we would still have to avoid the information leaks discussed in Section 4.1 and support Secure Logging.

In a cold-boot attack [14], an attacker manages to extract the symmetric key used for decrypting location information from the physical memory of the Trusted Module. In our architecture, we use a symmetric key only once for decrypting encrypted location information and then replace it with a fresh key, based on a cryptographic hash of the old key. The private key used for transferring the initial symmetric key to the Trusted Module is erased immediately after the transfer. These precautions give us perfect forward secrecy. Perfect backward secrecy is less of a concern. An attacker would have to steal the current symmetric key, the TLS private key, and other state information from the physical memory and set up a fake Trusted Module, without the network operator noticing that the original Trusted Module has been powered off. To make this attack even harder, the Trusted Module can periodically send signed, timestamped keep-alive messages to the network operator. If these messages are delayed by more than a threshold, the network operator stops sending location information to the module and requests its reboot, resulting in a new attestation operation.

9 Evaluation

In this section, we study the processing times of service queries and the feasibility of timing attacks.

9.1 Query Processing Times

We have two goals when studying the processing times of service queries. First, we want to measure overall response times for the nearby-friends and the interesting-places services. Second, we want to identify bottlenecks. We do not study the cost of remote attestation; this has been done before [36], and it occurs during initialization of the Trusted Module, so it is not part of the critical path.

We use the following setup to evaluate our architecture: We deploy the Trusted Module as discussed in Section 7. For simplicity reasons, we put all the other modules and the two databases on a single desktop machine with an Intel Pentium 4 (3 GHz) processor. The communication between the desktop machine and the Trusted Module runs over a wired network. For the nearby-friends service, the user and her friend each have a BlackBerry 8820 device, which has WiFi and GPS. Both devices periodically inform the network operator of their location; the user also transmits her location when issuing a service query. The devices communicate over WiFi. For the interesting-places service, we do not use a BlackBerry device, instead we use a slow laptop (a ThinkPad T43 with a 2 GHz Intel Pentium M processor) to approximate the capabilities of a modern smartphone. The laptop reports the location of its access point as its location to the network operator. While the BlackBerry devices are able to perform asymmetric cryptographic operations efficiently, as required by the interesting-places service, reporting these measurements raises licensing issues with Certicom, the provider of these operations.

For both the nearby-friends and the interesting-places service, we execute ten service queries. In case of the second service, we have the Places Database return 100 places to the Trusted Module, which then marks ten of them as valid. Our results are shown in Table 3, we report overall response time and the processing times of a service query and its response by the various modules. We report 'n/a' when a cost is found to be negligible. For the nearby-friends service, the average response time is about 107 ms. We find that most of the cost is due to performing private-key operations (signing and decrypting for Forwarder Module, signing for Trusted Module), where a single operation takes about 19 ms on the desktop machine and 35 ms on the T60 ThinkPad. The delay caused by communicating over WiFi instead of issuing a service query directly on the desktop machine is in the order of 15 ms.

	Nearby Friends [ms]		Interesting Places [ms]	
Cellphone (query)	n/a		62	(9)
Forwarder Module (query)	24	(5)	n/a	
Query Processor Module (query)	n/a		19	(1)
Trusted Module	37	(1)	80	(2)
Query Processor Module (response)	n/a		n/a	
Forwarder Module (response)	21	(2)	19	(1)
Cellphone (response)	n/a		44	(0)
Overall response time	107	(3)	213	(3)

Table 3: Query processing times. We present mean and standard deviation (in parentheses).

Machine	Code example 1				Code example 2			
	Negative		Positive		Negative		Positive	
Desktop	484	(4)	555	(1)	4,008	(14)	4,080	(15)
Laptop	517	(1)	561	(0)	4,980	(9)	5,034	(8)

Table 4: Cycle counts for code examples in Section 4.1.3. We present mean and standard deviation (in parentheses).

For the interesting-places services, the response time increases to about 213 ms. Here, the T43 ThinkPad signs the service query and decrypts the response. As expected, the T43 ThinkPad is slower than the other two computers. The Forwarder Module no longer needs to sign the service query, but it still needs to decrypt the response for filtering purposes. The processing times of the Trusted Module and of the Query Processor Module increase, which is due to the cost of receiving records from the Places Database and processing them. Note that the overall response time is shorter than the individual processing times, because some of the operations are executed in parallel. (E.g., the laptop can asymmetrically decrypt the symmetric key that was used for encrypting the places while the Forwarder Module is still filtering them.)

In summary, the measurements show that the processing delay of our architecture is hardly noticeable by a user. If the cost of the asymmetric cryptographic operations is a concern, downgrading to 1024 bit RSA keys will still provide adequate security in the medium term.

9.2 Timing Attacks

In the next experiment, we study the feasibility of timing attacks, as explained in Section 4.1.3. Our implementation does not allow a service provider to take measurements directly on the platform of the Trusted Module, instead the provider has to perform a remote timing attack, whose feasibility has been demonstrated by Brumley and Boneh [3].

We use the "rdtsc" instruction to retrieve the current value of the CPU cycle counter before and after executing a piece of code. We precede each "rdtsc" instruction with a "cpuid" instruction, which serializes the processor and avoids interference from out-of-order execution. Therefore, the absolute running time includes the execution time for one "cpuid" instruction, but we care only about relative running time.

We perform our measurements on the T60 ThinkPad and the desktop machine. We run four experiments on each machine, an experiment is repeated 100 times. In the first two experiments, we measure the running time of the single line of code required for determining whether a friend is nearby, as shown in Section 4.1.3.

In the first experiment, the variables are chosen such that the answer is negative and in the second one such that the answer is positive. The third and fourth experiments consist of 100 iterations of the second example show in Section 4.1.3. For example, for the interesting-places service, this code could be used to infer which of 100 places returned from the Places Database are in a user's area of interest. In the third experiment, the answer is negative for all the places, in the fourth one, it is always positive, which is a worst-case scenario. We present our results in Table 4.

From our results, we conclude that there is a statistically significant difference between the execution times of the positive and the negative case for both code examples. However, the difference is at most in the order of 80 cycles, which is multiple orders of magnitudes smaller than the differences required by Brumley and Boneh for a successful *remote* timing attack (thousands or millions, depending on the expected error). Furthermore, as shown in Table 3, the standard deviation of the processing time for the Trusted Module is in the order of milliseconds, so the differences in the cycle counts disappear in this noise. Finally, Secure Logging prevents a service provider from actively mounting a timing attack. Instead, the provider has to passively observe service queries and their responses, which limits the number of available measurements.

In summary, our measurements show that remote timing attacks are very unlikely to be successful.

10 Related Work

Research in location privacy has studied how to address the apparent contradiction, where, on the one hand, a location-based service needs access to location information to be useful, but, on the other hand, it should not have this access because of privacy concerns. A popular approach is location cloaking [4, 7, 11, 13, 15, 30], where only coarse-grained, less intrusive location information is given to a location-based service. Other work [2, 32] has suggested the usage of pseudonyms to keep a person's identity hidden from a service. However, some location-based services do require fine-grained location information or information about the person's identity, else service quality could suffer [4]. Services that alert parents when their child leaves a boundary area or that alert people of nearby friends fall into this category. Our architecture supports such services.

Kölsch et al. [24] also examine location privacy in a setup where the network operator and the service provider are different entities. They introduce a trusted third party that tracks users and that alerts the service provider when the user is in a particular area. The authors do not present an implementation of their concept, in particular, they do not discuss who acts as this third party. Our Trusted Module, as introduced in Section 3, can be looked at as such a third party. Apart from having a proof-of-concept implementation, our architecture has the additional advantage that it can exploit identity information and fine-grained location information, which increases service quality, whereas the previous work relies on coarse-grained location information and pseudonyms. Ravi et al. [35] have a service provider migrate the code that implements a location-based service to a network operator. The operator uses information flow control to ensure that the code does not leak a cellphone's fine-grained location to the provider. This approach is targeted at services that exploit aggregate location information and does not support services that require fine-grained location, as supported by our solution.

The usage of Trusted Computing or similar technologies for privacy protection has been proposed in earlier work. Molnar et al. [31] give a secret required for reading RFID tags to a reader only if the reader passes a remote configuration check. There is no implementation of this idea and no discussion of how the operator of a reader might indirectly gain access to data read by the reader. Iliev and Smith [21] deploy a tamperproof coprocessor at a server and use it for implementing a Private Information Retrieval (PIR) scheme [5], which allows database accesses without the owner of the database learning which records are being accessed. We could use PIR to let a cellphone retrieve information about interesting places close to its location without the service provider operating the database learning the location. However, Iliev and

Smith's scheme calls for periodic re-shuffles of all the records in the database, which is expensive. In terms of (single-server) PIR schemes that do not require a tamperproof coprocessor, Sion and Carbunar [38] show that these schemes are less time-efficient than simply transferring the entire database. Ghinita et al. [10] argue that the latter approach is not in the interest of a database owner and study the performance of a single-server, computational PIR scheme in the context of an interesting-places service. In many of the scenarios studied by the authors, the security of the PIR scheme relies on the difficulty of factoring a modulus of length 768 bits, which is far below the minimum length of 1024 bits recommended by NIST [1]. For a modulus of length 1024 bits, the authors find that each query requires at least ten seconds of server processing time. Moreover, the response to a query lets a client client learn $\sqrt{n}$ out of n interesting places stored in the database, so for the maximum value of n studied by the authors, $n = 100,000$, a client can learn the entire database by asking only 317 queries. In summary, our compromise, as explained in Section 4.1.1, is more efficient and leaks no unnecessary information to a client, but reveals coarse-grained location information to the service provider.

Previous research has explored the usage of Trusted Computing technologies to validate the configuration of a remote server [26, 27, 28, 36]. Here, a client ensures that the server (e.g., a web or SSH server) corresponds to an approved version, that it has not been tampered with, or that it does not leak a secret (e.g., a password). Our problem is harder because of the presence of the service provider, who can passively and actively intercept queries/responses flowing into/out of the service platform and measure their timing, which might leak location information (see Section 4.1). Moreover, to limit the danger of an exploitable vulnerability being present in the configuration trusted by the client, we do not include an entire server in the validated configuration, as opposed to Marchesini et al. [26] and Sailer et al. [36], only the part that directly deals with location information. McCune et al. [27, 28] study how to reduce this trusted computing base by exploiting a dynamic root of trust, but this solution currently has high per-query overhead (see Section 8). Finally, we give an explicit set of requirements that need to be satisfied in order for a particular version of software getting approval, which is a topic that has been neglected in previous work.

In terms of Secure Logging, Schneier and Kelsey [37] look at logging in the presence of an attacker. However, their threat model is different from ours. They are concerned about an attacker breaking into a machine, like a service provider managing to subvert our architecture, and modifying existing log entries. This is not a threat that we study in this paper, so the previous research is orthogonal to ours. A problem that we, but not the previous work, study is how to securely initialize the logging process upon a reboot of the service platform.

11 Conclusions and Future Work

We have demonstrated that it is possible to build efficient location-based services for which the service provider does not become aware of users' fine-grained location. In addition to keeping location information away from a service provider by encrypting it, we have also addressed several other passive and active attacks that a service provider might perform to gain access to this information.

Future work involves studying solutions based on a dynamic root of trust and applying our approach to protect the privacy of personal information other than location information, not necessarily only in location-based services.

Acknowledgments

We thank Research in Motion for providing us with two BlackBerry 8820 devices. This work is supported by the Natural Sciences and Engineering Research Council of Canada.

References

[1] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management - Part 1: General (Revised). NIST Special Publication 800-57, May 2006.

[2] A. R. Beresford and F. Stajano. Location Privacy in Pervasive Computing. *IEEE Pervasive Computing*, 2(1):46–55, 2003.

[3] D. Brumley and D. Boneh. Remote Timing Attacks are Practical. In *Proceedings of 12th USENIX Security Symposium*, August 2003.

[4] R. Cheng, Y. Zhang, E. Bertino, and S. Prabhakar. Preserving User Location Privacy in Mobile Data Management Infrastructures. In *Proceedings of 6th Workshop on Privacy Enhancing Technologies (PET 2006)*, pages 393–412, June 2006.

[5] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private Information Retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[6] Cybit Ltd. mapAmobile. `http://www.mapamobile.com`. Accessed June 2008.

[7] M. Duckham and L. Kulik. A Formal Model of Obfuscation and Negotiation for Location Privacy. In *Proceedings of 3rd International Conference on Pervasive Computing (Pervasive 2005)*, pages 152–170, May 2005.

[8] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. Van Randwyk, and D. Sicker. Passive Data Link Layer 802.11 Wireless Device Driver Fingerprinting. In *Proceedings of 15th USENIX Security Symposium*, July/August 2006.

[9] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Towards Trustworthy Kiosk Computing. In *Proceedings of 8th IEEE Workshop on Mobile Computing Systems and Applications (HotMobile'07)*, pages 41–45, February 2007.

[10] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K.-L. Tan. Private Queries in Location Based Services: Anonymizers are not Necessary. In *Proceedings of 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD'08)*, pages 121–132, June 2008.

[11] G. Ghinita, P. Kalnis, and S. Skiadopoulos. PRIVÉ: Anonymous Location-Based Queries in Distributed Mobile Systems. In *Proceedings of 16th International World Wide Web Conference (WWW2007)*, pages 371–380, May 2007.

[12] Google. Google Maps Mobile. `http://www.google.com/gmm`. Accessed June 2008.

[13] M. Gruteser and D. Grunwald. Anonymous Usage of Location-Based Services Through Spatial and Temporal Cloaking. In *Proceedings of 1st International Conference on Mobile Systems, Applications, and Services (MobiSys 2003)*, pages 31–42, May 2003.

[14] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proceedings of 17th USENIX Security Symposium*, July/August 2008.

[15] T. Hashem and L. Kulik. Safeguarding Location Privacy in Wireless Ad-Hoc Networks. In *Proceedings of 9th International Conference on Ubiquitous Computing (UbiComp 2007)*, pages 372–390, September 2007.

[16] U. Hengartner. Hiding Location Information from Location-Based Services. In *Proceedings of International Workshop on Privacy-Aware Location-based Mobile Services (PALMS)*, May 2007.

[17] U. Hengartner. Location Privacy based on Trusted Computing and Secure Logging. In *Proceedings of 4th International Conference on Security and Privacy in Communication Networks (SecureComm 2008)*, September 2008.

[18] IBM. IBM PCI-X Cryptographic Coprocessor. `http://www-03.ibm.com/security/cryptocards/pcixcc/overhardware.shtml`. Accessed November 2007.

[19] IBM. TrouSerS: The open-source TCG Software Stack. `http://trousers.sourceforge.net`. Accessed November 2007.

[20] IBM Research. Integrity Measurement Architecture (IMA). `http://sourceforge.net/projects/linux-ima`. Accessed June 2008.

[21] A. Iliev and S. W. Smith. Protecting Client Privacy with Trusted Computing at the Server. *IEEE Security and Privacy*, 3(2):20–28, March/April 2005.

[22] T. Jiang, H. J. Wang, and Y.-C. Hu. Preserving Location Privacy in Wireless LANs. In *Proceedings of 5th International Conference on Mobile Systems, Applications, and Services (MobiSys 2007)*, pages 246–257, June 2007.

[23] B. Kauer. OSLO: Improving the security of Trusted Computing. In *Proceedings of 16th USENIX Security Symposium*, pages 229–237, August 2007.

[24] T. Kölsch, L. Fritsch, M. Kohlweiss, and D. Kesdogan. Privacy for Profitable Location Based Services. In *Proceedings of 2nd International Conference on Security in Pervasive Computing (SPC 2005)*, pages 164–178, April 2005.

[25] U. Kühn, M. Selhorst, and C. Stüble. Realizing Property-Based Attestation and Sealing with Commonly Available Hard- and Software. In *Proceedings of 2nd ACM Workshop on Scalable Trusted Computing (STC'07)*, pages 50–57, November 2007.

[26] J. Marchesini, S. W. Smith, O. Wild, J. Stabiner, and A. Barsamian. Open-Source Applications of TCPA Hardware. In *Proceedings of 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 294–303, December 2004.

[27] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An Execution Infrastructure for TCB Minimization. In *Proceedings of ACM European Conference on Computer Systems (EuroSys 2008)*, March/April 2008.

[28] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and A. Seshadri. Minimal TCB Code Execution. In *Proceedings of IEEE Security and Privacy Symposium*, pages 267–272, May 2007.

[29] J. M. McCune, A. Perrig, and M. K. Reiter. Bump in the Ether: A Framework for Securing Sensitive User Input. In *Proceedings of USENIX Annual Technical Conference*, pages 185–198, May 2006.

[30] M. F. Mokbel, C.-Y. Chow, and W. G. Aref. The New Casper: Query Processing for Location Services without Compromising Privacy. In *Proceedings of 32nd International Conference on Very Large Data Bases (VLDB 2006)*, pages 763–774, September 2006.

[31] D. Molnar, A. Soppera, and D. Wagner. RFID Privacy Through Trusted Computing. In *Proceedings of Workshop on Privacy in the Electronic Society (WPES'05)*, November 2005.

[32] G. Myles, A. Friday, and N. Davies. Preserving Privacy in Environments with Location-Based Applications. *Pervasive Computing*, 2(1):56–64, January-March 2003.

[33] J. Pang, B. Greenstein, R. Gummadi, S. Seshan, and D. Wetherall. 802.11 User Fingerprinting. In *Proceedings of 13th International Conference on Mobile Computing and Networking (MobiCom 2007)*, September 2007.

[34] M. Prabaker, J. Rao, I. Fette, P. Kelley, L. Cranor, J. Hong, and N. Sadeh. Understanding and Capturing People's Privacy Policies in a People Finder Application. In *Proceedings of 5th Workshop on Privacy in UbiComp*, September 2007.

[35] N. Ravi, M. Gruteser, and L. Iftode. Non-Inference: An Information Flow Control Model for Location-based Services. In *Proceedings of 3rd International Conference on Mobile and Ubiquitous Systems: Networks and Services (MobiQuitous 2006)*, July 2006.

[36] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of 13th USENIX Security Symposium*, August 2004.

[37] B. Schneier and J. Kelsey. Cryptographic Support for Secure Logs on Untrusted Machines. In *Proceedings of 7th USENIX Security Symposium*, pages 53–62, January 1998.

[38] R. Sion and B. Carbunar. On the Computational Practicality of Private Information Retrieval. In *Proceedings of 14th Annual Network and Distributed System Security Symposium (NDSS 2007)*, February/March 2007.

[39] Trusted Computing Group. `https://www.trustedcomputinggroup.org`. Accessed June 2008.

[40] WaveMarket, Inc. Family Finder. `http://www.wavemarket.com`. Accessed June 2008.

[41] World-Tracker.Com. `http://www.world-tracker.com`. Accessed June 2008.

[42] G. Zhong, I. Goldberg, and U. Hengartner. Louis, Lester and Pierre: Three Protocols for Location Privacy. In *Proceedings of 7th Privacy Enhancing Technologies Symposium (PET 2007)*, pages 62–76, June 2007.