Privacy-Preserving Location Tracking of Lost or Stolen Devices: Cryptographic Techniques and Replacing Trusted Third Parties with DHTs

Thomas Ristenpart[*] *Gabriel Maganis*[†] *Arvind Krishnamurthy*[†] *Tadayoshi Kohno*[†]

[*]*University of California, San Diego*
tristenp@cs.ucsd.edu

[†]*University of Washington*
{gym,arvind,yoshi}@cs.washington.edu

Abstract

We tackle the problem of building *privacy-preserving device-tracking systems* — or private methods to assist in the recovery of lost or stolen Internet-connected mobile devices. The main goals of such systems are seemingly contradictory: to hide the device's legitimately-visited locations from third-party services and other parties (*location privacy*) while simultaneously using those same services to help recover the device's location(s) after it goes missing (*device-tracking*). We propose a system, named Adeona, that nevertheless meets both goals. It provides strong guarantees of location privacy while preserving the ability to efficiently track missing devices. We build a version of Adeona that uses OpenDHT as the third party service, resulting in an immediately deployable system that does not rely on any single trusted third party. We describe numerous extensions for the basic design that increase Adeona's suitability for particular deployment environments.

1 Introduction

The growing ubiquity of mobile computing devices, and our reliance upon them, means that losing them is simultaneously more likely and more damaging. For example, the annual CSI/FBI Computer Crime and Security Survey ranks laptop and mobile device theft as a prevalent and expensive problem for corporations [16]. To help combat this growing problem, corporations and individuals are deploying commercial *device-tracking* software — like "LoJack for Laptops" [1] — on their mobile devices. These systems typically send the identity of the device and its current network location (e.g., its IP address) over the Internet to a central server run by the device-tracking service. After losing a device, the service can determine the location of the device and, subsequently, can work with the owner and legal authorities to recover the device itself. The number of companies offering such services, e.g., [1, 9, 21, 29, 34, 37, 38], attests to the large and growing market for device tracking.

Unfortunately, these systems are incompatible with the oft-cited goal of *location privacy* [17, 22, 23] since the device-tracking services can always monitor the location of an Internet-enabled device — even while the device is in its owner's possession. This presents a significant barrier to the psychological acceptability of tracking services. To paraphrase one industry representative: companies will deploy these systems in order to track their devices, but they won't like it. The current situation leaves users of mobile devices in the awkward position of *either* using tracking services *or* protecting their location privacy.

We offer an alternative: *privacy-preserving device-tracking systems*. Such a system should provide strong guarantees of location privacy for the device owner's legitimately visited locations while nevertheless enabling tracking of the device after it goes missing. It should do so even while relying on untrusted third party services to store tracking updates.

The utility of device tracking systems. Before diving into technical details, we first step back to reevaluate whether device tracking, let alone privacy-preserving device tracking, even makes sense as a legitimate security tool for mobile device users. A motivated and sufficiently equipped or knowledgeable thief (i.e., the malicious entity assumed in possession of a missing device) can *always* prevent Internet device tracking: he or she can erase software on the device, deny Internet access, or even destroy the device. One might even be tempted to conclude that the products of [1, 9, 21, 29, 34, 37, 38] are just security "snake oil".

We purport that this extreme view of security is inappropriate for device tracking. While device tracking will not always work, these systems *can* work, and vendors (who may be admittedly biased) claim high recov-

[*]Work done while at the University of Washington.

ery rates [1]. The common-case thief is, after all, often opportunistic and unsophisticated, and it is against such thieves that tracking systems can clearly add significant value. Our work aims to retain this value while simultaneously addressing the considerable threats to user location privacy.

System goals. A device tracking system consists of: client hardware or software logic installed on the device; (sometimes) cryptographic key material stored on the device; (sometimes) cryptographic key material maintained separately by the device owner; and a remote storage facility. The client sends *location updates* over the Internet to the remote storage. Once a device goes missing, the owner or authorized agent searches the remote storage for location updates pertaining to the device's current whereabouts.

To understand the goals of a privacy-preserving tracking system, we begin with an exploration of existing or hypothetical tracking systems in scenarios that are derived from real situations (Section 2). This reveals a restrictive set of deployment constraints (e.g., supporting both efficient hardware and software clients) and an intricate threat model for location privacy where the remote storage provider is untrusted, the thief may try to learn past locations of the device, and other outsiders might attempt to glean private data from the system or "piggyback" on it to easily track a device. We extract the following main system goals.

(1) Updates sent by the client must be *anonymous* and *unlinkable*. This means that no adversary should be able to either associate an update to a particular device, or even associate two updates to the same (unknown) device.

(2) The tracking client must ensure *forward-privacy*, meaning a thief, even after seeing all of the internal state of the client, cannot learn past locations of the device.

(3) The client should protect against *timing attacks* by ensuring that the periodicity of updates cannot be easily used to identify a device.

(4) The owner should be able to efficiently search the remote storage in a privacy-preserving manner.

(5) The system must match closely the efficiency, deployability, and functionality of existing solutions that have little or no privacy guarantees.

These goals are not satisfied by straightforward or existing solutions. For example, simply encrypting location updates before sending to the remote storage does not allow for efficient retrieval. As another example, mechanisms for generating secure audit logs [32], while seemingly applicable, in fact violate our anonymity and unlinkability requirements by design.

We emphasize that one non-goal of our system is *improved* device tracking. As discussed above, all tracking systems in this category have fundamental limitations. Indeed, our overarching goal is to show that, in any setting where deploying a device tracking system makes sense, one can do so effectively *without compromising privacy*.

Adeona. Our system, named Adeona after the Roman goddess of "safe returns," meets the aggressive goals outlined above. The client consists of two modules: a location-finding module and a cryptographic core. With a small amount of state, the core utilizes a forward-secure pseudorandom generator (FSPRG) to efficiently and deterministically encapsulate updates, rendering them anonymous and unlinkable, while also scheduling them to be sent to the remote storage at pseudorandomly determined times (to help mitigate timing attacks). The core ensures forward-privacy: a thief, after determining all of the internal state of the client and even with access to all data on the remote storage, cannot use Adeona to reveal past locations of the device. The owner, with a copy of the initial state of the client, can efficiently search the remote storage for the updates. The cryptographic core uses only a sparing number of calls to AES per update.

The cryptographic techniques in the Adeona core have wide applicability, straightforwardly composing with any location-finding technique or remote storage instantiation. We showcase this by implementing Adeona as a fully functional tracking system using a public distributed storage infrastructure, OpenDHT [30]. We could also have potentially used other distributed hash table infrastructures such as the Azureus BitTorrent DHT. Using a DHT for remote storage means that there is no single trusted infrastructural component and that deployment can proceed immediately in a community-based way. End users need simply install a software client to enable private tracking service. Our system provides the first device tracking system not tied to a particular service provider. Moreover, to the best of our knowledge, we are also the first to explore replacing a centralized trusted third-party service with a decentralized DHT.

Extensions. Adeona does make slight trade-offs between simplicity, privacy, and device tracking. We address these trade-offs with several extensions to the basic Adeona system. These extensions serve two purposes: they highlight the versatility of our basic privacy-enhancing techniques and they can be used to better protect the tracking client against technically sophisticated thieves (at the cost of slight increases in complexity). In particular, we discuss several additions to the basic functionality of Adeona. For example, we design a novel cryptographic primitive, a tamper-evident FSPRG, to allow detection of adversarial modifications to the client's state.

Implementation and field testing. We have implemented the Adeona system and some of its extensions as user applications for Linux and Mac OS X. Moreover, we conducted a short trial in which the system was deployed on real users' systems, including a number of laptops. Our experience suggests that the Adeona system provides an immediate solution for privacy-preserving device tracking. The code is currently being readied for an open-source public release to be available at `http://adeona.cs.washington.edu/`, and we encourage the further use of this system for research purposes.

Outline. In the next section we provide a detailed discussion of tracking scenarios that help motivate our (involved) design constraints and threat models. Readers eager for technical details might skip ahead to Section 3, which describes the Adeona core. The full system based on OpenDHT is given in Section 4. We provide a security analysis in Section 5. Our implementations, their evaluation, and the results of the field trial appear in Section 6. We discuss Adeona's suitability for further deployment settings in Section 7 and extensions to Adeona are detailed in Section 8. We conclude in Section 9.

2 Problem Formulation

To explore existing and potential tracking system designs and understand the variety of adversarial threats, we first study a sequence of hypothetical tracking scenarios. While fictional, the scenarios are based on real stories and products. These scenarios uncover issues that will affect our goals and designs for private device tracking.

Scenario 1. Vance, an avid consumer of mobile devices, recently heard about the idea of "LoJack for Laptops." He searches the Internet, finds the EmailMe device tracking system, and installs it on his laptop.[1] The EmailMe tracking client software sends an email (like the example shown in Figure 1) to his webmail account every time the laptop connects to the Internet. Months later, Vance is distracted while working at his favorite coffee shop, and a thief takes his laptop. Now Vance's foresight appears to pay off: he uses a friend's computer to access the tracking emails sent by his missing laptop. Working with the authorities, they are able to determine that the laptop last connected to the Internet from a public wireless access point in his home city. Unfortunately the physical location was hard to pinpoint from just the IP addresses. A month after the theft Vance stops receiving tracking emails. An investigation eventually reveals that the thief sold the laptop at a flea market to an unsuspecting customer.[2] That customer later resold the laptop at a pawn shop. The pawnbroker, before further reselling the laptop, must have refurbished the laptop by wiping its hard drive and installing a fresh version of the operating system.

Discussion: The theft of Vance's laptop highlights a few issues regarding limitations on the functionality of device tracking systems. First, a client without hardware-support can provide network location data only when faced by such a *flea-market attack*: these occur when a technically unsophisticated thief steals a device to use it or sell it (with its software intact) as quickly as possible. Second, network location information will not always be sufficient for precisely determining the *physical location* of a device. Third, all clients (even those with hardware support) can be disabled from sending location updates (simply by disallowing all Internet access or by filtering out just the location updates if they can be isolated).

The principal goal of this paper is not to achieve better Internet tracking functionality than can be offered by existing solutions. Instead, we address privacy concerns while maintaining device tracking functionality equivalent to solutions with no or limited privacy guarantees. The next scenarios highlight the types of privacy concerns inherent to tracking systems.

Scenario 2. A few weeks before the theft of Vance's laptop, Vance was the target of a different kind of attack. His favorite coffee shop had been targeted by crackers because the shop is in a rich neighborhood and their routers are not configured to use WPA [19]. The crackers recorded all the coffee shop's traffic, including Vance's location-update emails, which were not encrypted. (The webmail service did not use TLS, nor does the EmailMe client encrypt the outgoing emails.) The crackers sell the data garnered from Vance's tracking emails to identity thieves, who then use Vance's identity to obtain several credit cards.

Discussion: The content of location updates should always be sent via *encrypted channels*, lest they reveal private information to passive eavesdroppers. This is of particular importance for mobile computing devices, because of their almost universal use of wireless communication, which may or may not use encryption.

Scenario 3. Vance works as a salesman for a small distributor of coffee-related products, called Very Good Coffee (VGC). He recently went on a trip abroad for VGC to investigate purchasing a supplier of coffee beans. On his return trip, he was stopped at customs and his laptop was temporarily confiscated for an "inspection" [28, 33]. Vance, with his ever-present foresight, had predicted this would happen: he encrypted all his sensitive work-related files and removed any information that might leak what he had been doing while in country. The laptop was shortly returned with files apparently unmodified.

Unknown to Vance, the EmailMe client had cached

```
From:  tech@brigadoonsoftware.com
To:  tech@brigadoonsoftware.com
BCC: tomrist@gmail.com
Subject:  Information

PCPH Pro For Win 95/98/ME/NT/2K/XP - Version 3.0 (Eval)

Date:  16-08-2007
Time:  11:14:05
Computer Name :  TOM-8F760D01401
User Name :  LOCAL SERVICE
IPAddress :0.0.0.0
IPAddress :128.208.7.80
Mac Address:  00-18-8B-A2-05-E5
Mac Address:  00-18-DE-9B-F0-5A
Serial Number:  DC44BF26
Registrants Name:  Tom
Organization:  Tom
Address:  513 Brooklyn Avenue
City:  Seattle
State/Province:  WA
Zip/Postal Code:  98105
Country:  USA
Work Phone:  2066163997
...
```

Figure 1: Example tracking email sent (unencrypted) by PC Phone Home [9] from one of the authors' laptops.

all the recently visited network locations on the laptop. Included were several IP addresses used by the supplier that VGC intended to purchase. The customs agents sold this information to a local competitor of VGC. Using this tip, the local competitor successfully blocked VGC's bid to purchase the supplier.

Discussion: This scenario addresses the need for *forward privacy*. A tracking client should not cache previous locations, lest a thief (or even, as the scenario depicts, some other untrusted party with temporary access to the device) easily break the owner's past location privacy.

Scenario 4. Hearing about Vance's recent troubles with property and identity theft, the VGC management chose to contract with (the optimistically named) All Devices Recovered (AllDevRec) to provide robust tracking services for VGC's mobile assets. AllDevRec, having made deals with laptop manufacturers, ensures that VGC's new laptops have hardware-supported tracking clients installed. The clients send updates using a proprietary protocol over an an encrypted channel to AllDevRec's servers each time an Internet connection is made.[3]

Ian, a recovery-management technician employed by AllDevRec, has a good friend Eve who happens to work at a business that competes with VGC. Ian brags to Eve that his position in AllDevRec allows him to access the locations from which VGC's employees access the Internet. This gives Eve an idea, and so she goads Ian into giving her information on the network locations visited by VGC sales people. From this Eve can infer the coffee shops VGC is targeting as potential customers, allowing her company to precisely undercut VGC's offerings.

Discussion: Using encrypted channels is insufficient to guarantee data privacy once the location updates reach a service provider's storage systems. The location updates should remain *encrypted while stored*. This mitigates the level of trust device owners must place in a service provider's ability to enforce proper data management policies (to protect against insider attacks) and security mechanisms (to protect against outsiders gaining access).

Scenario 5. Vance, now jobless due to VGC's recent bankruptcy, has been staying at Valerie's place. Valerie works at a large company, with its own in-house IT staff. The management decided to deploy a comprehensive tracking system for mobile computing asset management. To ensure employee acceptability of a tracking system, the management had the IT staff implement a system with privacy and security issues in mind: each device is assigned a random identification number and a public key, secret key pair for a public-key encryption scheme. The database mapping a device to its identification number, public key, and secret key is stored on a system with several procedural safeguards in place to ensure no unwarranted accesses. With each new Internet connection, the tracking client sends an update encrypted under the public key and indexed under the random identification number.

When Valerie goes to lunch (which varies in time quite a bit depending on her work), she heads across the street to a cafe to get away from the office. She often uses her company laptop and the cafe's wireless to peruse the Internet. Since deployment of the new tracking system, Valerie has been complaining that no matter when she takes lunch, Irving (a member of the IT staff who is reputed to have an unreciprocated romantic interest in her) almost always ends up coming by the cafe a few minutes after she arrives.[4]

Because the location updates sent by Valerie's laptop use a static identifier, it was easy for Irving (even without access to the protected database) to infer which was hers: he looked at identifiers with updates originating from the block of IP addresses used within Valerie's department and those used by the cafe. After a few guesses (which he validated by simply seeing if she was at the cafe), Irving determined her device's identification number and from then on knew whenever she went for lunch.

Discussion: The use of unchanging identifiers (even if originally anonymized) allows *linking attacks*, in which an adversary observing updates can associate updates from different locations as being from the same device.

Additionally, the finely-grained timing information revealed by sending updates upon each new Internet connection is a side-channel that can leak information.

Summary. The sequence of scenarios depicts the wide variety of potential users of tracking systems. Moreover, they highlight two fundamental security goals.

- Vance was a victim of compromised *device tracking*. (Scenario 1.)
- Vance, VGC, and Valerie were all victims of compromised *location privacy*. (Scenarios 2, 3, 4, and 5.)

The threat models related to achieving location privacy while retaining device tracking capabilities are complex because there exist numerous adversaries with widely varied powers and motivation:

- The unscrupulous party in possession of a device, which we will simply call the *thief*. The thief might be unsophisticated, sophisticated and intent on disabling the tracking device, or sophisticated and wish to reveal past locations.
- Internet-connected *outsiders* that might intercept update traffic (e.g., the crackers at the coffee shop). Such adversaries call for ensuring the use of encrypted channels.
- The *remote storage provider*, or the entity controlling the system(s) that host location updates, might be untrustworthy, suggesting the need for location updates that are *anonymous, unlinkable, and encrypted*, thereby denying private information even to the remote storage provider.

3 The Adeona Core: Providing Anonymous, Unlinkable Updates

The core module is the portion of a client primarily responsible for preparing, scheduling, and sending location updates to the remote storage. The Adeona core is, consequently, the foundation of our tracking system's privacy properties. We treat its development first, and mention that the core stands by itself as a component that will work in numerous deployment settings, in addition to the setting handled by the full Adeona system (described in the next section).

The discussion in Section 2 illustrates that the Adeona core must provide mechanisms to

(1) ensure content sent to the remote storage is anonymous and unlinkable;

(2) ensure forward-privacy (stored data on the client should not be sufficient for revealing previous locations);

(3) mitigate timing attacks; and

(4) allow the owner to efficiently search the remote storage for updates.

Basic design. A first approach for building a core would be to just utilize a secure symmetric encryption scheme. That is, the owner could install on the client a secret key and also store a copy separately, perhaps printed on a piece of paper or stored on a secure removable token. For each new Internet connection, the core would encrypt the location data using this secret key and immediately send the ciphertext to the remote storage. Goal (1) above would be satisfied because (assuming one used a standard, secure encryption scheme) these ciphertexts would, indeed, be anonymous and unlinkable. But, the other three goals are *not* met. A thief that gets access to the device and the secret key could decrypt previous updates. Sending the ciphertext immediately upon detecting a new Internet connection also leaks fine-grained timing information. More importantly, since ciphertexts submitted by all users are anonymous, there is no efficient way for the owner to search the database for his updates.[5]

The Adeona core utilizes a more sophisticated approach to tackle the other goals while preserving the ability to address goal (1). Instead of a key for an encryption scheme, the owner initializes the client with a *secret cryptographic seed* for a pseudorandom generator (PRG) [6]. Each time the core is run it uses the PRG and the seed to deterministically generate two fresh pseudorandom values: an index and a secret key (for the encryption scheme). The location information is encrypted using the secret key. The core sends both the index and the ciphertext to the remote storage. As before the ciphertext reveals no information, but the index is pseudorandom as well, meaning the entire update is anonymous and unlinkable. Thus goal (1) is satisfied. Goal (4) is as well: the owner, having a copy of the original cryptographic seed, can recompute all of the indices and keys used. This allows for efficient search of the remote storage for his or her updates, using the indices. The indices do not reveal decryption keys nor past or future indices.

This approach does not yet satisfy goal (2), because a thief — or customs official — can also use the seed to generate all the past indices and keys. We can rectify this by using a *forward-secure pseudorandom generator* (FSPRG) [5]: instead of using a single cryptographic seed for the lifetime of the system, the core also evolves the seed pseudorandomly. When run, the core uses the FSPRG and the seed to generate an index, secret key, and a new seed. The old seed is discarded (securely erased). The properties of the FSPRG ensure that it is computationally intractable to "go backwards" so that previous seeds (and the associated indices and keys) remain unknown even to a thief with access to the current seed.

Finally we can address goal (3) by randomly select-

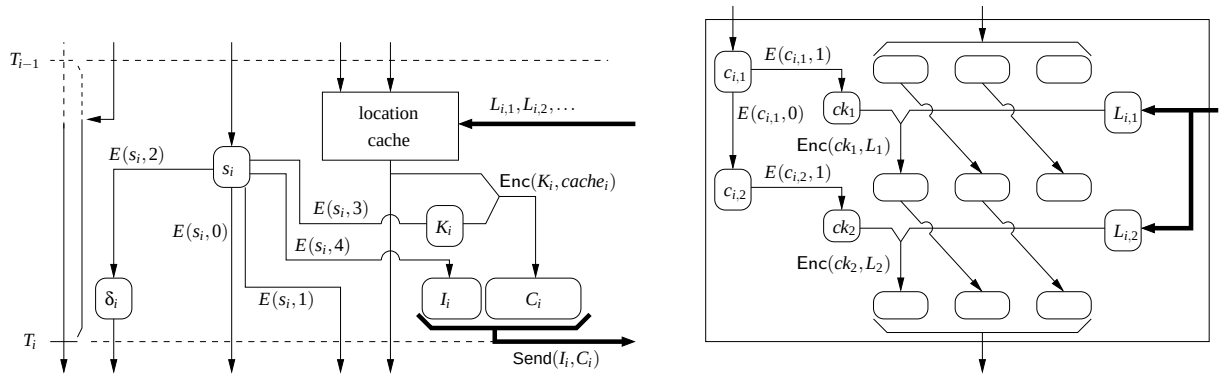


Figure 2: **(Left)** The Adeona core, where E is a block cipher (e.g., AES) instantiating the FSPRG and Enc is a standard encryption scheme. **(Right)** Close-up of the core's forward-private location caching, where the cache holds 3 updates and shown are two new locations being stored.

ing times to send updates. Using the FSPRG as a source of randomness, we can pseudorandomly generate exponentially-distributed inter-update times. (This allows the owner to also recompute the inter-update times, which will be useful for retrieval as discussed in Section 4.) Such a distribution is memoryless, meaning that, from the storage provider's view, the next update is equally likely to come from any client. We can tune the number of updates sent by adjusting the rate of the exponential distribution used.

Forward-private location caching. Our pseudorandom update schedule means that we might miss locations that are visited for only a short amount of time. However, to provide maximal evidentiary forensic data about the trajectories of a device after theft, we would like the core to allow reporting all of the recently visited locations. We could cache recent locations, but this breaks forward-privacy. We therefore enhance the basic design to include a *forward-private location cache*. Having a cache also provides a simple mechanism for adding temporal redundancy to updates (i.e., location data is sent multiple times to the remote storage over time), which can increase the ability to successfully retrieve updates.

Instead of just caching location data in the clear, we can have the core immediately encrypt new data sent from the location-finding module. The resulting ciphertext can then be added to a cache; the least recent ciphertext is expelled. However, we cannot just utilize the encryption key generated by the current state's FSPRG: a thief could decrypt any ciphertexts in the cache that were added since the last time the FSPRG seed was refreshed (e.g., when the previous update was sent). We therefore use a distinct FSPRG seed, which we call the *cache seed*, as the source for generating encryption keys for each location encountered. Each time the cache seed is used to encrypt new location data, it is also used to generate a new cache seed and the prior one is securely erased. In this way we guarantee forward privacy: no data in the core allows a thief to decrypt previously generated ciphertexts. When its time to send an update, the entire cache is encrypted using the secret key generated by the FSPRG with the main seed. This (second) encryption ensures that the data stored at the remote storage cannot later be correlated with ciphertexts in the cache. Finally, the core "resets" the cache seed by generating a fresh one using the FSPRG and the main seed. This associates a sequences of cache seeds to a particular update state. We ensure freshness of location data by mandating that at least one newly generated ciphertext is included with each update submitted to the remote storage.

The owner can reconstruct all of the cache seeds for any state (using the prior state's main seed) and do trial decryption to recover locations. (The number of expected trials is the number of locations visited in between two updates, and so this will be typically small.) Ciphertexts in the cache that are "leftover" from a prior update time period can also be decrypted, and this can be rendered efficient if plaintexts include a hint (i.e., the number of states back) that specifies which state generated the keys for the next ciphertext entry.

Implementing the design. Implementing the Adeona core is straightforward, given a block cipher[6] such as AES. A standard and provably secure FSPRG implementation based on AES works as follows [5]. A cryptographic seed is just an AES key (16 bytes). To generate a string of pseudorandom bits, one iteratively applies AES, under key a seed s, to a counter: $\mathsf{AES}(s,1)$, $\mathsf{AES}(s,2)$, etc. For Adeona, we have an initial main seed s_1 and initial cache seed $c_{1,1}$ (both randomly generated). The main seed s_1 is used to generate a new seed $s_2 = \mathsf{AES}(s_1,0)$, the next state's cache seed $c_{2,1} = \mathsf{AES}(s_1,1)$, and so on for the encryption key, index, and time offset. (The exponentially distributed time offset is generated from a pseudorandom input using the well known method of

inverse-transform sampling [13].) A seed, after it is used, must be securely erased. The cache seed forms a separate branch of the FSPRG and is used to generate a sequence of cache seeds and intermediate encryption keys for use within the cache. Figure 2 provides a diagram of the core module's operation between two successive updates at times T_{i-1} and T_i.

The encryption scheme can also be built using just AES, via an efficient block cipher mode such as GCM [26]. Such a mode also provides authenticity. Of added benefit is that the mode can be rendered deterministic (i.e., no randomness needed) since we only encrypt a single message with each key. This means that the core (once initialized) does not require a source of true randomness.

Summary. To summarize, the core uses a sequence of secret seeds $s_1, s_2, \ldots$ to provide

- a sequence $I_1, I_2, \ldots$ of *pseudorandom indices* to store ciphertexts under,
- sequences $c_{i,1}, c_{i,2}, \ldots$ of *secret cache seeds* for each state i that are then used to encrypt data about each location visited,
- a sequence $K_1, K_2, \ldots$ of *secret keys* for encrypting the cache before submission to the remote storage, and
- a sequence $\delta_1, \delta_2, \ldots$ of *pseudorandom inter-update times* for scheduling updates

while providing the following assurances. Given any I_i, K_j, or δ_l, no adversary can (under reasonable assumptions) compute any of the other output values above. Additionally, even if the thief views the entire internal state of the core, it still cannot compute any of the core's previously used indices, cache seeds, encryption keys, or inter-update times.

4 The Adeona System: Private Tracking using OpenDHT

A (privacy-preserving) tracking system consists of three main components: the device, the remote storage; and an owner. The device component itself consists of a location-finding component and a core component; other components — such as a camera image capture functionality — can easily be incorporated. A system works in three phases: initialization, active use, and retrieval. We have already seen the Adeona core. In this section we show how to construct a complete privacy-preserving device tracking system using it.

Our target is to develop an open-source, immediately deployable system. This will allow evaluation of our techniques during real usage (see Section 6), not to mention providing to individual users an immediate (and, to our knowledge, first) alternative to the plethora of existing, proprietary tracking systems, none of which achieve the level of privacy that we target and that we believe will be important to many users. Along these lines, this section focuses on a model for a open source software-only client. We use the public distributed storage infrastructure OpenDHT [30] for the remote storage facility. Not only does this obviate the need to setup dedicated remote storage facilities, enabling immediate deployability, but it effectively removes our system's reliance on any single trusted third party. This adds significantly to the practical privacy guarantees of the system.

We now flesh out the design of the complete Adeona system. The client consists of the Adeona core of the previous section (with a few slight modifications described below) plus a location-finding module, described below. First, however, we describe the other components: using OpenDHT for remote storage and how to perform privacy-preserving retrieval. We conclude the section with a summary of the whole system.

OpenDHT as remote storage. A distributed hash table (DHT) allows insertion and retrieval of data values based on hash keys. OpenDHT is an implementation of a distributed hash table (DHT) whose nodes run on PlanetLab [11]. We use the indices generated by the Adeona core as the hash keys and store the ciphertext data under them. There are several benefits to using a public, open-source distributed hash table (DHT) as remote storage. First, existing DHT's such as OpenDHT are already deployed and usable, meaning deployment of the tracking system only requires distribution of software for the client and for retrieval. Second, a DHT can naturally provide strengthened privacy and security guarantees because of the fact that updates will be stored uniformly across all the nodes of the DHT. In decentralized DHTs, an attacker would have to corrupt a significant fraction of DHT nodes in order to mount Denial-of-Service or privacy attacks as the storage provider.

On the other hand, DHT's also have limitations. The most fundamental is a lack of persistence guarantee: the DHT itself provides no assurance that inserted data can always be retrieved. Fortunately, OpenDHT ensures that inserted data is retained for at least a week.[7] Another limitation is temporary connectivity problems. Often nodes, even in OpenDHT, can be difficult to access, meaning our client will not be able to send an update successfully. The traditional approaches for handling such issues is to use client-side replication. This means that the client submits the same data to multiple, widely distributed nodes in the DHT.

We can enhance the Adeona core to include such a replication mechanism easily: have the core generate several indices (as opposed to just one) for each update. These indices, being pseudorandom already, will be distributed uniformly across the the space of all DHT nodes.

The update can then be submitted under all of these indices.

Scheduling location updates. The Adeona core provides a method to search for update ciphertexts via the deterministically generated indices. As noted, querying the remote storage for a set of indices does not reveal decryption keys or past or future indices. However, just the fact that a set of indices are queried for might allow the remote storage provider to trivially associated them to the same device. While the distributed nature of OpenDHT mitigates this threat, defense-in-depth asks that we do better. We therefore want a mechanism that ensures the owner can precisely determine which indices to search for when performing queries, and in particular allow him to avoid querying indices used before the device was lost or stolen.

To enable this functionality, we have the system precisely (but still pseudorandomly) schedule updates relative to some clock. The clock could be provided, for example, by a remote time server that the client and owner can synchronize against. Then, when the owner initializes the client, in addition to picking the cryptographic seed it also stores the current time as the initial time stamp T_1. Each subsequent state also has a time stamp associated with it: T_2, T_3, etc. These indicate the state's scheduled send time, and T_{i+1} is computed by adding T_i and δ_i (the pseudorandom inter-update delay). When the client is run, it reads the current time from the clock and iterates past states whose scheduled send time have already past. (In this way the core will "catch up" the state to the schedule.) With access to a clock loosely synchronized against the client's, the owner can accurately retrieve updates sent at various times (e.g., last week's updates, all the updates after the device went missing, etc.). We discuss the assumption of a clock more in our security analysis in Section 5.

Location-finding module. Our system works modularly with any known location finding technique (e.g., determining external IP address, trace routes to nearby routers, GPS, nearby 802.11 or GSM beacons, etc.). We implemented three different location-finding mechanisms: light, medium, and full. The *light mechanism* just determines the internal IP address and the externally-facing IP address. (The latter being the IP as reported by an external server.) The *medium mechanism* additionally performs traceroutes to 8 randomly-chosen PlanetLab nodes. These traceroutes provide additional information about the device's current surrounding network topology. The *full mechanism* employs a protocol that adapts state-of-the-art geolocationing techniques to our setting. Here, geolocationing refers to determining (approximate) physical locations from network data. Traditional approaches utilize a distributed set of landmarks to actively probe a target [18]. These probes, combined with the knowledge of the physical locations of the landmarks, allows approximate geolocationing of the target. We flip this approach around, using the active-client nature of our setting to have the client itself find nearby *passive landmarks*.

Concretely, we utilize Akamai [2] nodes as landmarks: they are numerous, widespread, and often co-located within ISPs (ensuring some node is usually very close to the device). Akamai is purported to have about 25 000 hosts distributed across 69 countries [2]. In a one-time pre-processing step, we can enumerate as many of their nodes as possible and then apply an existing virtual network coordinate system, Vivaldi [12], to assign them coordinates. The location-finding module chooses several nodes randomly out of this set, probes them to obtain round-trip times, then uses these values and the nodes' pre-computed virtual coordinates to determine the device's own virtual coordinates. Based on this, the module determines an additional set of landmarks that are close to it in virtual coordinate space and issues network measurements (pings and traceroutes) to these close landmarks. These measurements, in addition to the device's current internal- and external-facing IP addresses, are submitted to the core module as the current location information. After retrieval, this information can be used to geolocate the device, by potentially contacting the ISP hosting the edge routers.

Putting it all together. We describe the Adeona system in its entirety. A state of the client consists of the main cryptographic seed, the cache and its seed, and a time stamp. The main seed is used with an FSPRG to generate values associated to each state: the DHT indices, an encryption key, and an inter-update time. It also generates the next state's main seed and the next state's cache seed. The time stamp represents the time at which the current state should be used to send location information to the remote storage.

- (Initialization) The owner initializes the client by choosing random seeds and recording the time of initialization as the first state's time stamp. The cache is filled with random bits.
- (Active use) The main loop of the client proceeds as follows. The client, when executed, reads the current state and retrieves the current time (from, for example, the system clock). The client then transitions forward to the state that should be used to send the next update, based on the current time and the states' scheduled send times. The location cache uses its seed to appropriately encrypt each new location update received from the location module. At the scheduled send time, the main seed is used to generate several indices and an encryption key. The latter is used to encrypt the en-

tire cache. The result is inserted into OpenDHT under each index. The client then transitions to the next state. This means generating the next state's seed, the next state's cache seed, and the scheduled update time (the sum of the current update time and the inter-update delay). The old state data, except the cache, is erased.

- (Retrieval) To perform retrieval, the owner can use his or her copy of the initial state to recompute the sequence of states, their scheduled send times, and their associated indices and keys. From this information, he or she can determine the appropriate indices to search the remote storage (being careful to avoid indices from before the device went missing). After retrieving the caches, the owner can decrypt as described in Section 3.

5 Security Analysis

The Adeona system is designed to ensure location privacy, while retaining as much as possible the tracking abilities of solutions that provide weaker or no privacy properties. While we discuss other security evaluations and challenges inline in other sections, we treat here several key issues.

Location privacy. We discuss privacy first. We assume a privacy set of at least two participating devices, and do not consider omniscient adversaries that, in particular, can observe traffic at all locations visited by the device. (Such a powerful adversary can trivially compromise location privacy, assuming the device uses a persistent hardware MAC address.) The goal of adversaries is to use the Adeona system to learn more than their a priori knowledge about some device's visited locations. Because updates are anonymous and unlinkable, outsiders that see update traffic and the storage provider will not be able to associate the update to a device. The storage provider might associate updates that are later retrieved by the owner. This does not reveal anything about other updates sent by the owner's device. The randomized schedule obscures timing-related information that might otherwise reveal which device is communicating an update. Note also that the landmarks probed in our geolocationing module only learn that some device is probing them from an IP address. The thief cannot break the owner's location privacy due to our forward privacy guarantees.

Outsiders and the storage provider do learn that some device is at a certain location at a specific time (but not which device). Also, the number of devices currently using the system can be approximately determined (based on the rate of updates received), which could, for example, reveal a rough estimate of the number of devices behind a shared IP address. Moreover, these adversaries might attempt active attacks. For example, upon seeing an incoming update, the provider could immediately try to finger-print the source IP address [24]. Distributing the remote storage as with OpenDHT naturally makes such an attack more difficult to mount. There are also known preventative measures that mitigate a device's vulnerability to such attacks [24]. Finally, all of this could be protected against by sending updates via a system like Tor [14] (in deployment settings that would allow its use), which obfuscates the source IP address. See Section 8.4.

We remark that custom settings for Adeona's various parameters might reduce a device's privacy set. For example, if a client utilizes a cache size distinct from others, then this will serve to differentiate that client's updates. Likewise if a client submits more (or less) copies of each update to the remote storage, then the storage provider or outsiders might be able to differentiate its updates from those of other devices. Finally, a rate parameter significantly different from other clients' could allow tracking of the device.

Device tracking. We now discuss the goal of device tracking, which just means a system's ability to ensure updates about a missing device are retrieved by the owner. As mentioned previously, the goal here is for Adeona to engender the same tracking functionality as systems with weaker (or no) privacy guarantees. We therefore do not consider attacks which would also disable a normal tracking system: disabling the client, cutting off Internet access, destroying the device, etc. (Existing approaches to mitigating these attacks, like clever software engineering and/or hardware or BIOS support, are also applicable to our designs.) Nevertheless, Adeona as described in the previous section does have some limitations in this regard.

- OpenDHT does not provide everlasting persistence. This means that tracking fails for location updates more than a week old. Note that the location cache mechanism can be used to extend this time period. An alternate remote storage facility could also be used (see Section 7).
- Adeona schedules its updates at random times. If the device has Internet access for only a short time, this means that Adeona could miss a chance to send its update. We can trivially mitigate this by increasing the rate of our exponentially-distributed inter-update times (i.e., increase the frequency of updates), but at the cost of efficiency since this would mean sending more updates.
- The absolute privacy of retrieval relies on the device having a clock that the owner is loosely synchronized against. The client relies on the system clock to schedule updates. The thief could abuse this by,

for example, forcing the device's system clock to not progress. In the current implementation this would disrupt sending updates. Solutions for this are discussed in Section 8.1.

- Adeona relies on a stored state, and a thief could disable Adeona by tampering with it. For example, flipping even a single bit of the state will make all future updates unrecoverable. To ensure that the thief has to disable the client itself (and not just modify its state) we can use a tamper-evident FSPRG in conjunction with a "panic" mode of operation. See Sections 8.2 and 8.3.

For some of these bullets, we recall that many thieves will be unsophisticated. Therefore, in the common case the likelihood of the above attacks are small. (And, indeed, a sophisticated attacker could also compromise the tracking functionality of existing commercial, centralized alternatives.)

We also briefly mention that Adeona, like existing tracking systems, might not compose with some other mobile device security tools. For example, using a secure full-disk encryption system could render all software on the system unusable, including tracking software. We leave the question of how to securely combine tracking with other security mechanisms to future work.

Finally, while not a primary goal of our design, it turns out that Adeona's privacy mechanisms can actually *improve* tracking functionality. For one, the authentication of updates provided by our encryption mode means the owner knows that any received update was sent using the keys on the device, preventing in-transit tampering by outsiders or the storage provider. That updates are anonymous makes targeted Denial-of-Service attacks — in which the storage provider or an outsider attempts to selectively block or destroy an individual's updates — exceedingly difficult, if not impossible.

6 Implementation and Evaluation

To investigate the efficiency and practicality of our system, we have implemented several versions of the Adeona system as user-land applications for both Linux and Mac OS X. In all the versions, we used AES to implement the FSPRG. Encryption was performed using AES in counter mode and HMAC-SHA1 [3] in a standard Encrypt-then-MAC mode [4]. The OpenSSL crypto library[8] provided implementations of these primitives. We note that HMAC was used for convenience only; an implementation using AES for message authentication would also be straightforward. The rpcgen compiler was used to generate the client-side stubs for OpenDHT's put-get interface over the Sun RPC protocol. We also used Perl scripts to facilitate installation. We focus on three main versions.

- **adeona-0.2.1** implements the core functionality described in Sections 3 and 4. It uses the medium location-finding module of Section 4. The source code for adeona-0.2.1, not including the libraries mentioned above, consists of 7 091 lines of unoptimized C code. (Count includes comments and blank lines, i.e. calculated via wc -l *.[ch].) This version is being readied for public release.
- **adeona-0.2.0** is a slightly earlier version of adeona-0.2.1 that differs in that it uses a simpler version of the forward-private location cache. Its cache only handles locations observed during scheduled updates (as opposed to more frequent checks for a change in location, meaning that locations could be missed if ill-timed). The source code for adeona-0.2.0 consists of 5 231 lines of unoptimized C code. This version was deployed in the field trial described in Section 6.3.
- **adeona-0.1** uses the same ciphertext cache mechanism as adeona-0.2.0, and additionally includes the tamper-evident FSPRG that will be described in Section 8.2, the panic mode that will be described in Section 8.3, and the full location-finding mechanism described in Section 4. The tamper-evident FSPRG is implemented using the signature scheme associated to the Boneh-Boyen identity-based encryption (IBE) scheme [7] and the anonymous IBE scheme is implemented using Boneh-Franklin [8] in a hybrid mode with the Encrypt-then-MAC scheme described above. The two schemes rely on the same underlying elliptic curves that admit efficiently computable bilinear pairings. It relies on the Stanford Pairings-Based Crypto (PBC) library version 0.4.11 [25] and specifically the "Type F" pairings. Not counting the PBC library, this version is implemented in 9 723 lines of C code.

The oldest version was mainly for experimenting with the extensions discussed in Section 8 and the new geolocation technique discussed in Section 4, while the newer two versions were largely re-writes to prepare for public use. The source code for any version is directly available from the authors.

6.1 Performance

We ran several benchmarks to gauge the performance of our design mechanisms. The system hosting the experiments was a dual-core 3.20 GHz Intel Pentium 4 processor with 1GB of RAM. It was connected to the Internet via a university network.

Basic network operations. Table 2 gives the Wall-clock time in milliseconds (calculated via the gettimeofday system call) to perform each basic network operation: an OpenDHT put of a 1024-byte payload, an OpenDHT

	Min	Mean	Median	Max	T/O
Put	207	1 021	470	11 463	2
Get	2	240	77	11 238	3
Loc medium	5 642	13 270	15 531	30 381	–
Loc full	17 446	36 802	36 197	63 916	–

Table 1: Wall clock time in milliseconds/operation to perform basic network operations: DHT put, DHT get, a medium location-finding operation, and a full location-finding operation.

adeona-0.2.1	$r = 0$	$r = 10$	$r = 100$
Owner state	75	75	75
Client state (light)	75	876	8 076
Update (light)	36	400	4000
Client state (medium)	75	27 116	270 476
Update (medium)	1 348	13 520	135 200

adeona-0.1	$r = 0$	$r = 10$	$r = 100$
Owner state	3 544	3 545	3 548
Client state (full)	1 779	30 824	292 184
Update (full)	1 452	14 520	145 200

Table 2: Typical sizes in bytes of state and update data used by adeona-0.2.1 and adeona-0.1 on a 32-bit system, for different sizes of the ciphertext cache specified by r.

get of a 1024-byte payload, the time to do the 8 traceroutes used in the medium location-finding mechanism, and the time to do the full location-finding operation (as described in Section 4). Each operation was performed 100 times; shown is the min/mean/median/max time over the successful trials. The number of time outs (failures) for the put trials and get trials are shown in the column labeled T/O. The time out for OpenDHT RPC calls was set to 15 seconds in the implementation. For the location mechanisms, hop timeouts for traceroutes and timeouts for pings were set to 2 seconds (here an individual probe time out does not signify failure of the operation).

Space utilization. Table 2 details the space requirements in bytes of adeona-0.2.1 (adeona-0.2.0 has equivalent sizes) with light and medium location mechanisms and adeona-0.1 with the full location mechanism. Here, and below, the parameter r specifies the size of the ciphertext cache used. When $r = 0$ this means that no cache was used (only the current location is inserted during an update). For ease-of-use (i.e., so one can print out or copy down state information) we encoded all persistently stored data in hex, meaning the sizes of stored state are roughly twice larger than absolutely necessary. The use of asymmetric primitives by adeona-0.1 for the tamper-evident FSPRG functionality and the IBE scheme account for its larger space utilization.

Microbenchmarks. Space constraints limit the amount of data we can report, and so our focus here is on adeona-0.1. It uses more expensive cryptographic primitives (elliptic curves supporting bilinear pairings), and we want to assess whether the extensions relying on them hinder performance significantly. Table 3 gives running times in milliseconds/operation for the basic operations used by adeona-0.1. (We omit the times for non-panic encryption, decryption, update, and retrieve; these times were at most those of the related panic-mode operations.) These benchmarks only used the light location-finding mechanism and each update was inserted to a single OpenDHT node. Each operation was timed for 100 repetitions both using the clock system call (the CPU columns) and gettimeofday (the Wall columns); shown is the min/mean/median/max time over the successful trials. Where applicable, the number of time outs (due to DHT operations) are shown in the column labeled T/O. Note that the retrieve operations only include retrieval for a single update. These benchmarks show that the extensions are not prohibitive: performance is dependent almost entirely on the speed of network operations.

6.2 Geolocation accuracy

As mentioned earlier, our system has been designed to convey various kinds of location information to the storage service. We then rely on previously proposed network measurement analysis techniques and/or database lookups to process the stored location information and derive a geographical estimate. The strengths and weaknesses of such techniques are well-documented. We therefore focus our evaluation on the active client-based measurement technique described in Section 4 that attempts to identify a set of nearby passive landmarks given a large number of geographically distributed landmarks.

First, we accumulated about 225 412 open DNS servers by querying Internet search engines for dictionary words and collecting the DNS servers which responded to lookups on the resulting hostnames. Next, we enumerated 8 643 Akamai nodes across the world by querying the DNS servers for the IP addresses of hostnames known to resolve to Akamai edge servers (e.g., www.nba.com). Finally, 50 PlanetLab [11] nodes were used as stand-ins for lost or stolen devices across the United States.

Having both targets and landmarks, we obtained round-trip time (RTT) measurements from the PlanetLab nodes to the passive Akamai servers. The PlanetLab nodes were able to obtain measurements to 6 200 of the Akamai servers on our list. We could then evaluate our geolocation technique by running it over these measurements. Figure 3 plots the cumulative distributions of our results and the RTT to the actual closest Akamai node. We also plot the cumulative distribution of the RTT to an Akamai node as given by a simple DNS lookup for 32 of our 50 targets (the other 18 nodes went down at

Operation		CPU Min	CPU Mean	CPU Median	CPU Max	Wall Min	Wall Mean	Wall Median	Wall Max	T/O
Initialize core		210	329	330	460	215	367	348	1 082	–
Verify FSPRG state		340	456	470	610	351	494	474	1 240	–
Panic encryption		90	95	90	110	93	101	95	207	–
Panic decryption		80	90	90	100	85	104	90	934	–
Panic update	$r = 0$	440	559	570	700	612	1 653	977	15 347	9
	$r = 10$	440	543	545	680	818	2 289	1 311	20 582	10
	$r = 100$	540	666	690	800	2 953	12 599	7 439	165 950	5
Panic retrieve	$r = 0$	80	89	90	100	92	499	207	12 003	7
	$r = 10$	80	87	90	100	93	705	335	9 734	12
	$r = 100$	80	87	90	100	116	2 458	1 555	21 734	5

Table 3: Time in milliseconds to perform basic operations in adeona-0.1.

the time of measurement). Our technique performs better than Akamai's own content delivery algorithms for more than 60% of the the targets we considered. In addition, we observe that it can find an Akamai server at most 7 milliseconds away.

6.3 Field trial

We conducted a small field trial to gain experience with our implementation of Adeona, reveal potential issues with our designs, and quantitatively gauge the efficacy of using OpenDHT as a remote storage facility. There were 11 participants each running the adeona-0.2.0 client with the same options: update rate parameter of 0.002 (about 7 updates an hour on average), location cache of size $r = 4$, and spatial replication of 4 (the core tries to insert each update to 4 DHT keys). The clients were instrumented to locally log all the location updates submitted over the course of the trial. At the end of the trial, we collected these client-side log files plus each owner's copy of the initial state, and used this data to attempt to retrieve a week's worth of updates[9] for each of the participants.

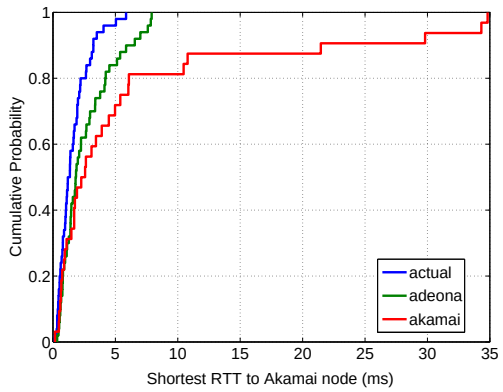
Results are shown in the left table of Figure 3. Here '# Inserts' refers to the total number of successful insertions into the DHT by the client in the week period. The 'Insert rate' column measures the fraction of these inserts that were retrieved. The '# Updates' column shows the total number of updates submitted by each client. Note that our replication mechanism means that each update causes the client to attempt 4 insertions of the location cache. The 'Update rate' column measures the percentage of location caches retrieved. As can be seen, this fraction is usually larger than the fraction of inserts retrieved, suggesting that replication across multiple DHT keys is beneficial. The 'Locations Found' column reports the number of unique locations (defined as distinct (internal IP, external IP) pair) found during retrieval versus reported. The final column measures the time, in minutes and seconds, that it took to perform retrieval for the user's updates for the whole week (note that we parallelized retrieval for each user).

We observed that OpenDHT may return "no data" for a key even when, in fact, there is data stored under that key. (This was detected when doing multiple get requests for a key.) Indeed, the failure to find two of the user locations was due to this phenomenon, and in fact repeating the retrieval operation found these locations as well.

7 Deployment Settings: Hardware Support and Dedicated Servers

In Section 2, we highlighted several settings for device tracking: internal corporate systems, third-party companies offering tracking services, and community-supported tracking for individuals. Each case differs in terms of what resources are available to both the tracking client and the remote storage. In Section 4 we built the Adeona system targeting a software client and OpenDHT repository, which works well for the third setting. Here we describe how our designs can work with other deployment scenarios.

A *hardware-supported* client can be deployed in several ways, including ASICs implementing client logic, trusted hardware modules (e.g., a TPM [35] or Intel's Active Management technology), or worked into existing system firmware components (e.g., a system BIOS). Hardware-support can be effectively used to ensure that the tracking client can only be disabled by the most sophisticated thieves and, possibly, that the client has access to a private and tamper-free state. However, targeting a system for use with a hardware-supported client *adds* to system requirements. While we do not work out all the (important) details of a hardware implementation of Adeona's client (leaving this to future work), we argue here that our techniques are amicable to this type of deployment. Adeona's core (Section 3) is particularly suited for implementation in hardware. It relies on a single cryptographic primitive, AES, which is highly optimized for both software and hardware. For example, recent research shows how to implement AES in only



User	# Inserts	Insert Rate	# Updates	Update Rate	Locations Found	Retrieve Time
01	491	0.89	251	0.94	11/12	12m 06s
02	632	0.89	327	0.94	3/3	16m 04s
03	622	0.84	321	0.91	2/2	17m 04s
04	543	0.87	274	0.95	5/5	15m 03s
05	617	0.88	309	0.96	4/4	19m 04s
06	234	0.85	123	0.90	4/4	15m 06s
07	359	0.89	199	0.95	5/6	18m 04s
08	420	0.85	220	0.92	7/7	14m 06s
09	504	0.91	259	0.97	1/1	11m 06s
10	138	0.90	59	0.92	4/4	13m 04s
11	302	0.81	175	0.91	6/6	14m 04s

Figure 3: **(Left)** The cumulative distribution of the shortest RTT (in milliseconds) to an Akamai node found by Adeona compared to the actual closest Akamai node and Akamai's own content delivery algorithm. **(Right)** Field trial retrieval rates and retrieval times (in minutes and seconds).

3400 gates (on a "grain of sand") [15]. In its most basic form (without a location cache), the core only requires 16 bytes of persistent storage.

In settings where a third-party company offers tracking services, the design requirements are more relaxed compared to a community-supported approach. Particularly, such a company would typically offer *dedicated remote storage servers*. This would allow handling persistence issues server-side. Further, this kind of remote storage service is likely to provide better availability than DHTs, obviating the need to engineer the client to handle various kinds of service failures. Adeona is thus slightly over-engineered for this setting (e.g., we could dispense with the replication technique of Section 4). An interesting question that is raised in such a deployment setting is how to perform privacy-preserving access control. For obvious reasons, these remote storage facilities would want to restrict the parties able to insert data. If we use traditional authentication mechanisms, the authentication tag might reveal who is submitting the update. Thus one might think about using newer cryptographic primitives such as group or ring signatures [10, 31] that allow authentication while not revealing what member of a group is actually communicating the update.

Corporations or other large organizations might opt to *internally host storage facilities*, as per Scenario 5 of Section 2. Again, dedicated storage servers ease design constraints, meaning Adeona can be simplified for this setting. But there is again the issue of access control. (Though in this setting existing corporate VPN's, if these do not reveal the client's identity, might be used.) On the other hand, this kind of deployment raises other interesting questions. Particularly, the privacy set is necessarily restricted to only employees of the corporation, and so an adversary might be able to aggregate information about overall employee location habits even if the adversary cannot track individual employees.

8 Extensions

We describe several extensions for the Adeona system that highlight its versatility and extensibility. These include: removing the reliance on synchronized clocks, tamper-evident FSPRGs for untrusted local storage, a panic mode of operation that does not rely on state, the use of anonymous channels, and enabling communication from an owner back to a lost device.

8.1 Avoiding synchronized time

The Adeona system, as described in Section 4, utilizes a shared clock between the client and owner to ensure safe retrieval. This is realized straightforwardly if the client is loosely synchronized against an external clock (e.g., via NTP [27]). In deployment scenarios where the device cannot be guaranteed to maintain synchronization or the thief might maliciously modify the system clock, we can modify the client and retrieval process as follows.

Whenever the client is executed, it reads the current state (which is now just the current cryptographic seed for the FSPRG and the cache) and computes the inter-update time δ associated to the state. It then waits that amount of time before sending the next update and progressing the state. For retrieval, the owner can still compute all of the inter-update times, and use these to estimate when a state was used to send an update. If the client runs continuously from initialization, then a state will be used when predicted by the sum of the inter-update times of earlier states. If the client is not run continuously from initialization, then a state might be used to send an update *later* (relative to absolute time) than predicted by the inter-update times. It is therefore

privacy-preserving for the owner to retrieve any states estimated to be sent after the time at which the device was lost. The owner might also query prior states to search for relevant updates, but being careful not to go too far back (lest he begin querying for updates sent before the device was lost).

8.2 Detection of client state tampering

The Adeona system relies on the client's state remaining unmodified. Compared to a (hypothetical) stateless client, this allows a new avenue for disabling the device. To rectify this disparity between the ideal (in which an adversary has to disable/modify the client executable) and Adeona we design a novel cryptographic primitive, a *tamper-evident FSPRG*, that allows cryptographic validation of state. By adding this functionality to Adeona we remove this avenue for disabling tracking functionality. Moreover, we believe that tamper-evident FSPRGs are likely of independent interest and might find use in further contexts where untrusted storage is in use, e.g., when the Adeona core is implemented in hardware but the state is stored in memory accessible to an adversary.

A straightforward construction would work as follows. The owner, during initialization, also generates a signing key and a verification key for a digital signature scheme. Then, the initialization routine generates the core's values $s_i, c_{i,1}, T_i$ for each future state i that could be used by the client, and signs these values. The verification key and resulting signatures are placed in the client's storage, along with the normal initial state. The client, to validate lack of tampering with FSPRG states, can verify the state's $s_i, c_{i,1}, T_i$ values via the digital signature's verification algorithm and the (stored) verification key. Unfortunately this approach requires a large amount of storage (linear in the number of updates that could be sent). Moreover, a very sophisticated thief could just mount a replacement attack: substitute his or her own state, verification key, and signatures for the owner's. Note this attack does not require modifying or otherwise interfering with the client executable. We can do better on both accounts.

To stop replacement attacks, we can use a trusted authority to generate a certificate for the owner's verification key (which should also tie it to the device). Then the trusted authority's verification key can be hard-coded in the client executable and be used to validate the owner's (stored) verification key. To reduce the storage space required, we have the owner, during initialization, only sign the *final* state's values. To verify, the client can seek forward with the FSPRG (without yet deleting the current state) to the final state and then verify the signature. (A counter can be used to denote how many states the client needs to progress to reach the final one.) Under reasonable assumptions regarding the FSPRG (in particular, that it's difficult to find two distinct states that lead to the same future state) and the assumption that the digital signature scheme is secure, no adversary will be able to generate a state that deviates from the normal progression, yet verifies. A clever thief might try to roll the FSPRG forward in the normal progression, to cause a long wait before the next update. This can be defended against with a straightforward check relative to the current time: if the next update is too far away, then assume adversarial modification. We could also store the signatures of some fraction of the intermediate states in order to operate at different points of a space/computation trade-off.

8.3 Private updates with tampered state

If the client detects tampering with the state, then it can enter into a "panic" mode which does not rely on the stored state to send updates. One might have panic mode just send updates in the clear (because these locations are presumably not associated with the owner), but there can be reasons not to do this. For example, configuration errors by an owner could mistakenly invoke panic mode.

Panic mode can still provide some protection for updates without relying on shared state, as follows. We assume the client and owner have access to an immutable, unique identification string ID. In practice this ID could be the laptop's MAC address. We also use a cryptographic hash function H: $\{0,1\}^* \rightarrow \{0,1\}^h$, such as SHA-256 for which $h = 256$ bits. Then pick a parameter $b \in [0..h]$. For each update, the client generates a sequence of indexes via $I_1 = H(1 \,||\, T \,||\, H(\mathsf{ID})|_b)$, $I_2 = H(2 \,||\, T \,||\, H(\mathsf{ID})|_b)$, etc. Here T is the current date and $H(\mathsf{ID})|_b$ denotes hashing ID and then taking the first b bits of the result. (Varying the parameter b enables a simple time-privacy trade off known as "bucketization".)

Location information can be encrypted using an anonymous identity-based encryption scheme [8]. Using a trusted key distribution center, each owner can receive a secret key associated to their device's ID. (Note that the center will be able to decrypt updates, also.) Encryption to the owner only requires ID. This is useful because then encryption does not require stored per-device state, under the presumption that ID is always accessible. The ciphertext can be submitted under the indices. The owner can retrieve these panic-mode updates by re-computing the indices using ID and the appropriate dates and then using trial decryption.

8.4 Anonymous channels

Systems such as Tor [14] implement anonymous channels, which can be used to effectively obfuscate from re-

cipients the originating IP address of Internet traffic. The Adeona design can easily compose with any such system by transmitting location updates to the remote storage across the anonymous channel. The combination of Adeona with an anonymous routing system provides several nice benefits. It means that the storage provider and outsiders do not trivially see the originating IP address, meaning active fingerprinting attacks are prevented. Additionally, it merges the anonymity set of Adeona with that of the anonymous channel system. For example, even if there exists only a single user of Adeona, that user might nevertheless achieve some degree of location privacy using anonymous channels.

On the other hand, attempting to use anonymous channels without Adeona does not satisfy our system goals. The now more complex clients would not necessarily be suitable for some deployment settings (e.g. hardware implementations). It would force a reliance on a complex, distributed infrastructure in all deployment settings. This reliance is particularly bad in the corporate setting. Routing location updates through nodes not controlled by the company could actually decrease corporate privacy: outsiders could potentially learn employee locations (e.g., see [36]). Moreover, when analyzing how to utilize anonymous channels and meet our tracking and privacy goals, it is easy to see that even with the anonymous channel one still benefits from Adeona's mechanisms. Imagine a hypothetical system based on anonymous channels. Because the storage provider is potentially adversarial, the system would still need to encrypt location information and so also provide an index to enable efficient search of the remote storage. Because the source IP is hidden, one might utilize a static, anonymous identifier. This would allow the storage provider to, at the very least, link update times to a single device, which leaks more information than if the indices are unlinkable.

8.5 Sending commands to the device

In situations where a device is lost, an owner might wish to not only retrieve updates from it, but also securely send commands back to it. For example, such a channel would allow remotely deleting sensitive data. We can securely instantiate a full duplex channel using the remote storage as a bulletin board. An owner could post encrypted and signed messages to the remote storage under indices of future updates. The client, during an update, would first do a retrieve on the indices to be used for the update, thereby receiving the encrypted and authenticated commands. Standard encryption and authentication tools can be used, including using cryptographic keys derived from the FSPRG seed in use on the client. In terms of location privacy, the storage provider would now additionally learn that two entities are communicating via the bulletin board, but not which entities.

9 Conclusion

This paper develops mechanisms by which one can build *privacy-preserving device tracking systems*. These systems simultaneously hide a device owner's legitimately visited locations from powerful adversaries, while enabling tracking of the device after it goes missing. Moreover, we do so while using third party services that are not trusted in terms of location privacy. Our mechanisms are efficient and practical to deploy. Our client-side mechanisms are well-suited for hardware implementations. This illustrates that not only can one circumvent a trade-off between security and privacy, but one can do so in practice for real systems.

We implemented Adeona, a full privacy-preserving tracking system based on OpenDHT that allows for immediate, community-orientated deployment. Its core module, the cryptographic engine that renders location updates anonymous and unlinkable, can be easily used in further deployment settings. To evaluate Adeona, we ran a field trial to gain experience with a deployment on real user's systems. Our conclusion is that our approach is sound and an immediately viable alternative to tracking systems that offer less (or no) privacy guarantees. Lastly, we also presented numerous extensions to Adeona that address a range of issues: disparate deployment settings, increased functionality, and improved security. The techniques involved, particularly our tamper-evident FSPRG, are likely of independent interest.

Notes

[1] EmailMe is a fictional system, though its functionality is based on products such as PC Phone Home [9] and Inspice [21].

[2] A flea market is a type of ad-hoc market where transactions are typically anonymous and done in cash.

[3] AllDevRec is a fictional company, though the services it offers are comparable to those advertised by Absolute Software [1], which has tracking software pre-installed in the BIOS of some Dell laptops.

[4] A real example of such insider abuse is found in [20].

[5] The owner could download the entire database and do trial decryption, but with many users this would be prohibitively expensive.

[6] One could also utilize as basic primitive a keyed hash function.

[7] To be precise, the guarantee is that OpenDHT guarantees not to expire a key-value pair before its time-to-live passes, barring some catastrophic failure of the DHT service [30].

[8] Systems we built on had version 0.9.7l or later. We used SHA1, instead of the more secure SHA-256, due to its lack of implementation in OpenSSL 0.9.7l (the most recent version available for OSX).

[9] To be precise, the search was for any update potentially sent over the course of 6 days and 23 hours. The final hour was omitted for simplicity since it avoided retrieving updates being expired by OpenDHT.

References

[1] Absolute Software. Computrace LoJack for Laptops. http://www.absolute.com/solutions-theft-recovery.asp

[2] Akamai website. http://www.akamai.com/

[3] M. Bellare, R. Canetti, and H. Krawczyk. Keying Hash Functions for Message Authentication. *CRYPTO*, 1996.

[4] M. Bellare and C. Namprempre. Authenticated-Encryption: Relations among notions and analysis of the generic composition paradigm. *ASIACRYPT*, 2000.

[5] M. Bellare and B. Yee. Forward-Security in Privatey-Key Cryptography. *CT-RSA*, 2003.

[6] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, vol. 13, no. 4, pp. 850–864, 1984.

[7] D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. In EUROCRYPT, 2004.

[8] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, 2001.

[9] Brigadoon Software, Inc. PC Phone Home. http://www.pcphonehome.com/

[10] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*, 1991.

[11] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An overlay testbed for broad-coverage services *ACM SIGCOMM*, 2003.

[12] F. Dabek, R. Cox, F. Kaashoek, and R. Morris Vivaldi: A decentralized network coordinate system. In *SIGCOMM*, 2004.

[13] L. Devroye. Non-Uniform Random Variate Generation. New York, Springer-Verlag, 1986.

[14] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. *USENIX Security Symposium*, USENIX, vol. 13, pp. 9–13, 2004.

[15] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *IEE Proc. Inf. Secur.*, vol. 152, no. 1, pp. 13–20, Oct. 2005.

[16] L. Gordon, M. Loeb, W. Lucyshyn, and R. Richardson. CSI/FBI Computer Crime and Security Survey 2006. Computer Security Institute publications, 2006.

[17] M. Gruteser and D. Grunwald. A Methodological Assessment of Location Privacy Risks in Wireless Hotspot Networks *Security in Pervasive Computing – SPC*, 2003.

[18] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of Internet hosts. *To appear in ACM Transactions on Networking.*

[19] IEEE Standards Association. IEEE Std 802.11i-2004. 2004.

[20] InformationWeek. Federal Agent Indicted for Using Homeland Security Database To Stalk Girlfriend. http://www.informationweek.com/shared/printableArticle.jhtml?articleID=201807903

[21] Inspice. Inspice Trace. http://www.inspice.com/

[22] M. Jakobsson and S. Wetzel. Security Weaknesses in Bluetooth. CT-RSA, 2001.

[23] A. Juels. RFID Security and Privacy: A Research Survey, *IEEE Journal on Selected Areas in Communications*, 2006.

[24] T. Kohno, A. Broido, and K.C. Claffy. Remote physical device fingerprinting. *IEEE Symposium on Security and Privacy*, 2005.

[25] B. Lynn. Stanford Pairings-Based Crypto Library. http://crypto.stanford.edu/pbc/.

[26] D. McGrew and J. Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. *INDOCRYPT*, 2004.

[27] D. Mills. Improved Algorithms for Synchronizing Computer Network Clocks. *IEEE/ACM Trans. Netw.*, vol. 3 no. 3, pp. 245–254, 1995.

[28] New York Times. At U.S. Borders, Laptops Have No Right to Privacy. http://www.nytimes.com/2006/10/24/business/24road.html . October 24, 2006.

[29] Raytheon Oakley Systems. SureFind. http://www.oakleynetworks.com/products/surefind.php

[30] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. In *SIGCOMM*, 2005.

[31] R. Rivest, A. Shamir, and Y. Tauman How to leak a secret. In *ASIACRYPT*, 2001.

[32] B. Schneier and J. Kelsey. Secure Audit Logs to Support Computer Forensics. ACM Transactions on Information and System Security, vol. 1, no. 3, 1999.

[33] Slashdot. US Courts Consider Legality of Laptop Inspection. http://yro.slashdot.org/article.pl?sid=08/01/08/1641209&tid=158 . January 8, 2008.

[34] Tri-8, Inc. MyLaptopGPS. http://www.mylaptopgps.com/

[35] Trusted Computing Group. http://www.trustedcomputinggroup.org/specs/TPM/

[36] Wired. Rogue Nodes Turn Tor Anonymizer Into Eavesdropper's Paradise. http://www.wired.com/politics/security/news/2007/09/embassy_hacks . September 10, 2007.

[37] XTool Mobile Security, Inc. XTool Laptop Tracker. http://www.xtool.com/xtooltracker.aspx

[38] zTrace Technologies. http://www.ztrace.com/