HAL
open science

Message Quality for Ambient System Security

Ciaran Bryce

To cite this version:

Ciaran Bryce. Message Quality for Ambient System Security. [Research Report] PI 1896, 2008, pp.20. inria-00300400

HAL Id: inria-00300400
https://inria.hal.science/inria-00300400v1

Submitted on 18 Jul 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

I R I S A
INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES

PUBLICATION
INTERNE
N° 1896

MESSAGE QUALITY FOR AMBIENT SYSTEM SECURITY

CIARÁN BRYCE

ISSN 1166-8687

IRISA
CAMPUS UNIVERSITAIRE DE BEAULIEU - 35042 RENNES CEDEX - FRANCE

INSTITUT DE RECHERCHE EN INFORMATIQUE ET SYSTÈMES ALÉATOIRES
Campus de Beaulieu – 35042 Rennes Cedex – France
Tél. : (33) 02 99 84 71 00 – Fax : (33) 02 99 84 71 71
http://www.irisa.fr

Message Quality for Ambient System Security

Ciarán Bryce*

Systèmes communicants
Projets ACES

Publication interne n° 1896 — July 2008 — 20 pages

Abstract: In ambient systems, a principal may be a physical object whose identity does not convey useful information for taking security decisions. Thus, establishing a trusted channel with a device depends more on the device being able to demonstrate *what it does*, rather than *who it is*. This paper proposes a security model that allows a principal to establish the intent of an adversary and to make the adversary prove its trustworthiness by furnishing proof of current and past behavior.

Key-words: Security, ambient systems, trust, programming model.

(Résumé : tsvp)

* INRIA, Ciaran.Bryce@inria.fr

Centre National de la Recherche Scientifique
(UMR 6074) Université de Rennes 1 – Insa de Rennes

Institut National de Recherche en Informatique
et en Automatique – unité de recherche de Rennes

Un modèle de sécurité pour les systèmes diffus

Résumé : L'informatique diffuse est actuellement en plein essor. Les téléphones portables et autres dispositifs individuels sont de plus en plus répandus. Ce papier propose un modèle de sécurité pour ces environnements. Ce modèle repose sur la notion de confiance. Le principe retenu est que la confiance ne peut être établie que si une entité est capable de prouver que son comportement est conforme à celui qu'elle annonce à ses interlocuteurs.

Mots clés : Sécurité et confiance, modèle de programmation, informatique diffuse.

Contents

1 Introduction **4**

2 Security Challenges in Ambient Systems **5**

3 Message Quality: a Security Model **6**
3.1 Approach 6
3.2 Management of Message Quality 7

4 Model Implementation **11**
4.1 Prototype 11
4.2 Example 14
4.3 Analysis of Model 16

5 Related Work and Conclusions **18**

1 Introduction

The technological combination of portable devices (e.g., smart phones, PDAs), wireless networking, and processor cards embedded into everyday devices, has led to the emergence of *ambient computing systems*. Ambient systems are employed for process control and person-centric applications like mobile health [9], where devices monitor a person's vital functions and contact a doctor in the event of an emergency, domotics [13], where devices in building environments are used to monitor and control access to rooms, and payment schemes [10] where user devices hold purses or subscription information that is read each time user gains access to a service.

Information security is a major concern for ambient systems since personal devices can contain private user information and business subscription data, and run increasingly sensitive applications from the health and business domains. There are three major security risks that users of ambient systems face that can compromise the security of a device's information. One obvious risk is device theft, which is particularly serious if the device does not possess a means to protect the confidentiality of information that it stores from thieves. Another risk is virus and worm infections, especially since devices rely heavily on other devices in the network for services; increased network activity increases the risk of viral infections. Yet another risk is spam: already a pest for the Internet, spam seriously reduces the quality of a user's experience and wastes his network, storage and battery resources.

Implementing information security in ambient systems is quite an engineering challenge. For instance, devices that communicate can be unknown to each other, so the first step in communication is to establish mutual trust, or in other words, to set up a *trusted channel* between them. Since the network might use short-range radio, the network might not possess a trusted third party, like a certificate authority [16] or reputation server [18], that can help mistrusting devices establish a trusted channel. Thus, when two peers want to establish a trusted channel, the security model must ensure that there is sufficient security information already on their devices to establish the channel, and thus minimize reliance on third parties.

The goal of this paper is to investigate the security requirements for ambient systems and to propose a security model. This model is entitled *message quality* since its role is to enable a device to examine each message that it receives from a partner device, and to determine if that message is consequent to a security attack on, or by, the partner device. The model's implementation leverages support from the *Trusted Platform Module* (TPM) [19] – a general purpose hardware chip designed for secure computing that can be used by a platform to demonstrate that its software has not been tampered with.

A key feature of the security model is the deprecated role of principal identity: it can be more important for a device to prove *what it does* than it is to prove *who it is*. For instance, when a user PDA interacts with a soda vending machine, it is more useful for the PDA to establish that the machine returns sodas in return for payment than it is to know the vending machine's serial number. This approach requires that a security model validates the software running on the device, whereas traditional security models are designed to validate the identity of a principal. A second aim of the model is to enable principals to estimate

the trustworthiness of a partner principal, by being able to determine if their partner has behaved in a trustworthy way in the past.

This paper is organized as follows. Section 2 presents the security challenges of ambient computing systems that the paper addresses. The security model is presented in Section 3, and we comment on its strengths and weaknesses. In Section 4, the model is integrated into a programming model that is often used in ambient computing – the tuple space model [7]. An example of the security model in use is presented. Related work is presented in Section 5 which concludes the paper.

2 Security Challenges in Ambient Systems

Viruses remain one of the most common and most costly source of security attacks in information systems. Handheld devices are not immune to viruses, the Cabir virus for instance being the first major virus on the Symbian operating system and the Duts virus infecting PocketPCs[1]. A mobile device virus is potentially more pernicious than an Internet virus since a device can be engaged in a communication without the owner being aware. (One can always disconnect a wired communication link and thus be sure that the computer is not engaged in hidden communication).

A second major security concern in today's information systems is information misuse [6]. This is the problem of information being exposed or destroyed because of ineffective access controls to physical and information resources. A concrete example for handhelds is theft: the French interior ministry reports that up to 200 000 mobile phones are stolen in France each year[2].

A further risk for ambient systems is spam – a well known problem for the Internet, with over 70% of e-mail traffic consisting of unsolicited messages [11]. Spam is attractive for attackers (spamers) due to the low cost of sending messages. A similar situation can arise in ambient systems, where devices can send information to others at no cost – apart from the energy consumed by the device's battery. An example spam scenario is one where a user passes near a supermarket and picks up unwanted messages from items on sale.

We choose in this paper to concentrate on the above risks since they are relatively novel. There are of course other risks, like network blocking attacks, as well as traditional risks such as cracking the trusted hardware on devices and PIN theft.

There are a number of challenges to implementing a security infrastructure that can address these risks. These are:

- There is a potentially huge number of peers without centralized management. No peer knows all others, and most peers know few others. Strictly speaking, peer Alice *knows* Bob if she can link Bob's identity to his expected behavior. This lack of knowledge would normally imply the use of trusted third parties such as recommendation servers and certificate authorities. However, given the potential size of systems and weak

[1] http://www.virusthreatcenter.com/
[2] http://www.afom.fr/v3/TEMPLATES/acces_elus_l2.php?rubrique_ID=115

connectivity of wireless networks, trusted third party solutions must be minimized in favor of decentralized, scalable ones. In a decentralized solution, when Alice sends a message M to Bob, then the peers and message M already contain sufficient data for Bob to verify if it is safe to act upon the message, i.e., that the message is not consequent to a security attack.

- Personal computing devices have limited computing, storage and energy resources. Consequently, devices need to optimize network communications and general processing, and this restriction curtails the security infrastructure.
- Each device owner is autonomous and has complete control over his device. He is free to install any software on his device and gain access to any service. He can manipulate information on the device in any way, unless that data is stored in a trusted zone, possibly with the support of trusted hardware. A device may be stolen and then used or misused by a non-owner. A device owner might also be free to manage the identity of his device. This can lead to the so-called Sybil attack [5] where a malicious user changes the identity of his device and exploits this new identity to dissociate himself from his previous malicious behavior, by appearing as a new device with no record of malicious activity.
- Knowing a principal's name should not be considered the same thing as knowing its expected behavior or trust level. Systems in practice degrade over time through ware of hardware and software (as patches are applied for upgrades and virus/bug fixes). Embedded devices can degrade simply through physical ware to their environment. Thus, a hitherto trusted principal can start behaving badly, so a security infrastructure must be able to detect this.

3 Message Quality: a Security Model

This section introduces the message quality model. An overview of the model is given in Section 3.1 and the elements needed to implement the model are presented in Section 3.2.

3.1 Approach

There are two cornerstones to the security model presented here.

- The first is that a principal must prove *what it does* rather than *who it is* (i.e., its identity). A principal's identity might not change, but its ability to service a request – what it does – can change as new functionality is added, viruses spread, etc.
- The second cornerstone is decentralization. That is, as suggested in Figure 1, the security framework on a principal's device decides on the security of each message. If the message is acceptable, then it is passed up to higher application layers for processing.

Otherwise, the message gets rejected without the application being informed. For decentralization, there must be sufficient information in the message and pre-distributed to Bob to take this decision. This is necessary to allow a decision to be taken without real-time recourse to a (perhaps absent from network) trusted third party.

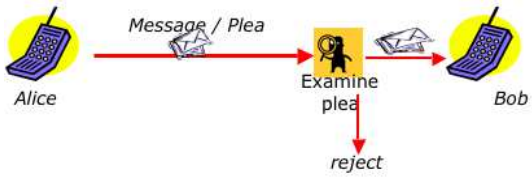

Figure 1: Message Quality: the sender provides a plea object that argues for the security of the message.

Consider a message M sent from Alice to Bob, c.f., Figure 1. There are three desirable properties of M that define its security and trustworthiness:

1. M **is plausible**. This is the property that M is a message that Alice is likely to utter, given the behavioral profile of Principal Alice. For instance, if Alice is a frequent taxi passenger, then the messages she is likely to utter include requests for a taxi. This property is useful in defending against masquerade attacks and helps to identify users with stolen devices.

2. M **is trustworthy**. This is the property that permits Bob to believe that the contents of message M are true. For example, if Alice sends a message asking for a taxi, then Bob – the taxi driver – can stop the taxi with sufficient confidence that there is a passenger waiting to mount. Trustworthiness is not an absolute value of a message. Rather, it is a feature whereby Alice can complement a message with proof that the contents of the message are trustworthy. This aspect leverages work done in trust-based frameworks, e.g., [18, 20]. In Figure 1, the proof is represented in the *plea* object.

3. M **is useful**. This is the property that Bob is interested in M. The message is rejected as spam by Bob's device if it does not correspond to the class of messages that Bob wishes to receive.

These properties are collectively known as *message quality* and, as is later argued, are crucial to addressing the risks outlined. Note how the onus is on the message sender to convince the receiver of the quality of its message.

3.2 Management of Message Quality

The abstractions used to implement message quality are **profiles**, **policies** and **pleas**. All are first-class objects in the message quality model.

A profile is a declarative entity that defines the expected behavior of a principal. Each principal carries a profile object and other principals can query it. A policy is a programmable entity held by each message receiver that is evaluated each time a message is received. The role of a policy is to determine if the message is sufficiently trustworthy, plausible and useful from the receiver's point of view. A plea is a copy of the sender's profile and history information that a sender includes in a message in order to argue for the message's quality.

Both profile and policy objects are declarative. They are installed on the device, perhaps at the same time as applications are installed. Since not all users will be competent in defining these objects, they could also be downloaded with an application. Installation of applications as well as profile and policy objects is a PIN-protected operation.

Principals The role of identity for principals is deprecated in the message quality model. While most principals do not need to furnish or even possess an identity, there is a small population of identities that are well-known. For instance, software providers (e.g., application and OS providers), service providers (e.g., a taxi company where passengers use their PDAs to gain access to the service) can be considered well-known since all passengers must interact with them at some stage, e.g., to download the software or renew their subscription. Another class of well-known principals are hardware providers, including the TPM manufacturers as well as the authorities that manage TPM cryptographic keys.

One role of the well-known principals can be to define programs and policies for devices since many users might not be in a position to define these for themselves.

Profiles Principal behavior is defined in a profile. A profile is qualified by the set of installed programs. A program, in turn, is qualified by the following information:

- The *actions* of the program. These are expressed as the set of messages that the program can send and receive.
- A certification of the program origin that describes where or by whom the program was developed. The certification of a program's origin can be denoted by a certificate. In this paper's model, this certificate is named the `createdBy` certificate.
- A certification of the program installation describing who installed the program on the device. This certification is denoted by the `installedBy` certificate.
- Any other application or service specific certifications that are considered useful in an application context, e.g., `inspectedBy`. Each device administrator is free to define his own certifications for program profiles.

The certificates in the profile are termed *profile certificates*. The role of a certificate is to bind a public key to a profile, which are keys belonging to well-known principals. We conjecture that the number of certificates is nonetheless minimized since they certify behavior, and not identity. For instance, there can be millions of taxi clients for a single taxi client behavioral

profile. Further, since the number of well-known principals is small, certificates can be pre-distributed, e.g., during software installation.

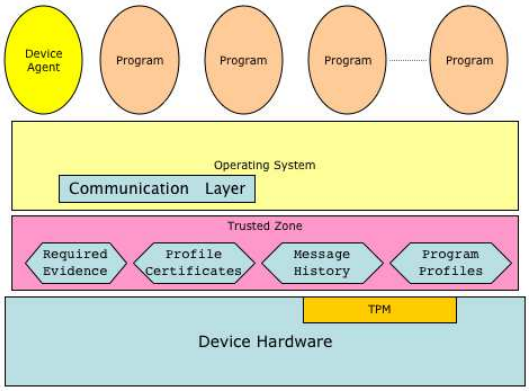

Figure 2: Ambient device environment

Policies When Bob receives a message, its security kernel rejects the message if it does not conform to the profile of Alice. Further, for utility, Bob can specify a *policy* on the acceptable profile of the message sender. A sender whose message profile does not conform with this policy has its message automatically rejected. The two elements of the policy are:

1. the *evidence* – a message history that the sender must demonstrate for trustworthiness, and,
2. the required profile certificates – specified as the role (i.e., createdBy, installedBy, etc.) and public key of well-known principals. Like programs, policies can be downloaded and installed on the device (since ordinary PDA users are not expected to understand the intricate details).

Platform The structure of an ambient device platform using the message quality model is illustrated in Figure 2. The OS or runtime environment runs alongside a software layer called the *trusted zone*. This zone stores profiles, policies and history data. The profile and policy can only be set by furnishing a PIN that, supposedly, only the owner of the platform knows. One of the programs running on the device is a device *agent*; it is a trusted program that interacts with well-known principals for downloading programs and profiles.

Verifying Message Quality A profile is used to construct a plea for each message sent. The other component of a plea is a *history* of messages sent and received by a principal. A plea sent along with a message M from Alice to Bob is used in the following way by Bob's trusted zone to verify message quality.

- Plausibility is verified by i): ensuring the M belongs to Alice's profile; ii) validating any profile certificates in the profile.
- Trustworthiness is verified by ensuring that Alice's history of messages matches a series of messages that Bob specifies as required evidence.
- Utility is verified by ensuring that M belongs to Bob's profile.

For precision, we formalize the core concepts of message quality verification. There are a basic number of data types τ_i used by principals in the messages they exchange, each with data values $\{ d_i \}$.

Each device stores a history of messages sent and received. This is defined as a sequence of data items D, where each entry is tagged as being a value received as input, i, or a value transmitted which is tagged with o.

$$H : \langle D^{i/o} \rangle$$

The profile of a device is defined as the sequence of message types received and sent by the device. An entry in a profile message sequence can be a value D or a type τ representing the fact that a value of that type gets sent or received.

$$P : \langle D|\tau^{i/o} \rangle$$

Similarly, part of a principal's policy is the required evidence E that an interacting principal must furnish with its message in order to demonstrate trustworthiness. This also is defined as a tuple of data or types.

$$E : \langle D|\tau^{i/o} \rangle$$

Profile certificates bind keys K to profiles for certain roles R (e.g., installedBy, createdBy, etc.):

$$C : R \rightarrow P \rightarrow K$$

A profile certificate c for a profile p is validated for a role r and public key k if the signature maps, i.e., $c(r, p) = k$.

A message m exchanged between two devices is a quadruple of a data item, a profile, a history and a list of profile certificates:

$$M : (D, P, H, C^*)$$

To verify message quality, a message m $(= (d, p_S, h, c^*))$ is analyzed against the receiving principal's policy. The policy is the required profile certificates (for role r_i and signature k_i), the required evidence e and his profile p_R. Verification examines plausibility (σ_P), utility (σ_U) and trustworthiness (σ_T), and also validates the profile certificates (σ_C).

$$
\begin{aligned}
\text{MessageQuality}[(r_i,\, k_i)^*,\, e,\, p_R](m) \;\hat{=}\; & \\
& \sigma_P[p_S](m) \quad (\hat{=} \quad m.d \sqsubseteq m.p_S) \\
& \wedge \\
& \sigma_T[e](m) \quad (\hat{=} \quad e \sqsubseteq m.h) \\
& \wedge \\
& \sigma_U[p](m) \quad (\hat{=} \quad m.d \sqsubseteq p_R) \\
& \wedge \\
& \sigma_C[c_i](m) \quad (\hat{=} \quad \forall .i\ c_i.(r_i,\, p_S) = k_i)
\end{aligned}
$$

The $\sqsubseteq$ operator represents the matching of a message data sequence D to a profile. For profile $P = \langle\, p_0,\, ...,\, p_n\, \rangle$, its length $len(P)$ is $n + 1$, where p_i is a data item $d^{i/o}$ or a type $\tau^{i/o}$, tagged with an input/output marker. A subsequence of a profile, with length k and starting from index j in m is defined as:

$$
subseq(P, j, k) = \begin{cases} \langle p_j, ..., p_{j+k} \rangle & \text{if } j + k \leq len(P) \\ undefined & j + k \quad > len(P) \end{cases}
$$

A match occurs if each data item $d_i^{i/o}$ in D has the same value and direction (input or output) as the corresponding entry in the profile, if this is a data value, or has the type specified for that entry in the profile.

$$
\exists j.\ subseq(P, j, len(D)).\begin{cases} p_i = d_i & \text{if } p_i \text{ is a data item} \\ p_i = \tau(d_i) & \text{if } p_i \text{ is a type} \end{cases}
$$

4 Model Implementation

This section presents implementation details of the model. A prototype is described in Section 4.1, and this is used to present an example in 4.2. Finally, Section 4.3 analyzes the security of the model.

4.1 Prototype

We implemented the tuple space framework in the Java programming language, this choice being facilitated by the widespread availability of Java environments for ambient computing devices and by the safety guarantees of the language. The latter are useful for ensuring that the trusted zone is not tampered with by user programs. The implementation is lightweight, containing only 3KLOC. Our motivation for implementing the message quality model is to experiment with its notion of security in a range of application environments.

The programming model chosen in this paper is based on the Linda tuple space model [7]. Communication in Linda is based on the exchange of typed data sequences called *tuples* via a message board (or *tuple space*). For example, `<"Taxi", "Airport", 30>` is a tuple where the first two elements are strings and the third is an integer. A tuple is placed in the tuple

space using the out operation, at which point it becomes visible to all processes or principals that have access to the tuple space. A tuple is addressed using patterns that match one or a set of tuples present in the tuple-space. An example pattern that matches the previous tuple is `<String, "Airport", int>`. The Linda operation to read this tuple from the tuple space is read(`<String, "Airport", int>`).

The tuple space model is well suited to ambient environments since a sending principal does not need to know the identity of the receiving principal and the receiver need not know the sender. This is useful since there might be no principal naming scheme in place. Further, the receiver need not be present in the network when the message is sent; unlike for socket communication, the receiver picks up the message when he is ready.

Our implementation is based on the Lana system [3], each principal has its own tuple space in which it publishes its tuples. The tuples in a principal's tuple space can be read by all other principals in its network vicinity; see Figure 3. Many systems designed for ambient environments employ Linda's tuple space model for the reasons cited above, e.g., Lime [14], Spread [4].

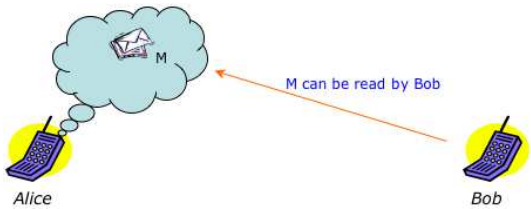

Figure 3: Each device has its own tuple space.

The list of operations is presented in Figure 3. The only change in semantics compared with the Linda variant is the read operation: if no device is present in the network, or if there is no matching tuple, the operation sleeps for some time and then tries again. The read operation does not remove the tuple from the space. The out operation publishes a tuple in the space of the current principal. As can be seen from Figure 3, the tuple space primitives do not contain explicit reference to the message quality model, so we consider its implementation in the programming model as transparent.

A key requirement for the message quality model is that a trusted zone be present on each device participating in the model. One approach to building trusted zones is to use software protection techniques, now possible using strongly typed languages like Java [8]. Another approach is to rely on the Trusted Platform Module (TPM) to enable a platform to demonstrate that the software it is running has not been tampered with. A TPM is a hardware device whose functionality is specified by the Trusted Computing Group [19]. The TPM is becoming commodity hardware, with 200 million TPM-enabled PCs having been shipped by the end of 2007.

A TPM is used to store measures of the platform, usually in the form of secure code and data hashes internally in its Platform Configuration Registers (PCRs). To verify that

Operation	Role
read(Tuple pattern)	Returns a tuple matching pattern from any device in network neighborhood
out(Tuple t)	Publishes t in the local tuple space
remove(Tuple t)	Removes t from local tuple space

Figure 4: Primitives of a tuple space model.

a device is running correct (non-modified) software, its TPM can be challenged to produce its stored PCR values signed using an Attestation Identity Key (AiK). This key is created by a TPM and certified by a well-known principal (called a *privacy authority*). The privacy authority that certifies the AiKs can be the application software provider. On contacting the provider, the client device downloads the correct digest values. As illustrated in Figure 5, when Alice sends a message m to Bob, this is in fact in reply to a tuple space query from Bob and a challenge to Alice's TPM. Bob can verify Alice's PCRs (and software) using his copy of the digest that he got when downloading the software. Trust is thus built in a transitive fashion: the digest permits Bob to believe that a correct version of the trusted zone is running on Alice's device, and then trust in Alice's trusted zone provider is sufficient for Bob to believe that Alice is correctly implementing the message quality model. The final step for building the trusted channel is then to verify message quality.

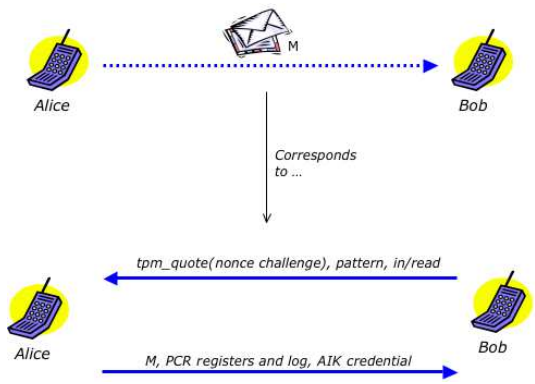

Figure 5: Message quality in tuple space model.

4.2 Example

The example illustrates a shuttle service that people can call using their PDAs. There are two classes of service: the fast service enables clients to call taxis from any location, the second requires them to go to a shuttle depot where they take the shuttle. The principal interactions are illustrated in Figure 6. The first three messages, for calling a shuttle, are only for fast shuttles; the final two message exchanges occur in all shuttles at the end of the trip. To call a shuttle, a customer sends a message –"Please" – with the desired destination. The shuttle replies with a fare quote, which the customer can accept by sending a "Stop" message. At the end of a trip, the shuttle sends an "Arrived" message to the customer, which is acknowledged by "Bye".

Figure 6: Shuttle scenario

There are two key security requirements that we want to implement in this application. One is plausibility for shuttles – customers can be sure that they are communicating with real shuttles, rather than with rogue devices masquerading as shuttles. The second requirement is customer trustworthiness: shuttles that receive requests need to believe that the request comes from a customer who wants to take a shuttle, rather than from someone playing customer messages "for fun". To increase trustworthiness, fast shuttles require that potential customers furnish evidence of previous shuttle rides. For trustworthiness, the taxi service assumes here that someone who has previously taken a taxi is less likely to lie about wanting to take the service again. The only inconvenience for customers is that their first ride via the service is on a slow shuttle.

The first extract, taken from a Java implementation, shows how the message exchange part of a profile (which is denoted *protocol*) can be simply defined. This is the profile of a fast shuttle and describes its behavior in relation to taking a customer. Recall, the principal

is unable to send messages that do not correspond to its profile since they get rejected by receiving principals due to message quality failure.

```
// Take a client protocol
TupleSequence.ExchangedTuple inc1, outc, inc2;
TupleSequence clientSeq = new TupleSequence();
inc1 = new InTuple(new Tuple(
   new Entry[]{new Entry.String("Please"),
      Entry.Type.Strings}));
outc = new OutTuple(new Tuple(
   new Entry[]{Entry.Type.Strings, Entry.Type.Ints}));
inc2 = new InTuple(new Tuple(
   new Entry[]{new Entry.String("Stop"),
      Entry.Type.Strings, Entry.Type.Ints}));
clientSeq = new TupleSequence(
   new TupleSequence.ExchangedTuple[]{inc1, outc, inc2});
fastTaxiProtocol.append(clientSeq);
```

When operating, the fast service shuttle accepts requests. Its first task is to define the evidence for servicing clients.

```
// Set up stuff -- in main()
TrustedZone tz = TrustedZone.getTrustedZone();
PIN pin = new PIN(111);
tz.setProfile(pin, TaxiProtocol.getFastTaxiProtocol());
tz.setRequiredEvidence(pin, TaxiProtocol.getEvidence());
tz.addRequiredCertificate(pin, Role.installedBy,
   Shuttle.pubKey);

while ( true ) {
    String destination = takeClient();
    handleReceipt(destination);
}

private static String takeClient() {
   String destination;
   // Detect passenger
   Entry te1, te2; Tuple t1;
   te1 = new Entry.String("Please");
   te2 = Entry.Type.Strings;
   t1 = TupleSpace.rd(new Tuple( new Entry[]{te1, te2} ));
   destination = ((Entry.String)t1.get(1)).extractString();
   // Give fare
   Entry te3, te4;
   te3 = new Entry.String(destination);
   te4 = new Entry.Int(30);
   TupleSpace.out(new Tuple(new Entry[]{te3, te4}));
   // Get OK from passenger
   Entry te5 = new Entry.String("Stop");
   TupleSpace.rd(new Tuple(new Entry[]{te5, te3, te4}));
   return destination;
}
```

```
private static void handleReceipt(String destination) {
   Entry te1, te2; Tuple t1, t2;
   te1 = new Entry.String("Arrived");
   te2 = new Entry.String(destination);
   t1 = new Tuple(new Entry[] { te1, te2 } );
   TupleSpace.out(t1);
   t2 = TupleSpace.rd(new Tuple(new Entry[]{ Entry.Any }));
```

The program starts by setting the profile of the principal. This can only be done by furnishing the correct PIN (which is something that a thief presumedly cannot know). The remainder of the code simply implements the protocol with the passenger. The call setRequiredCertificates specifies a profile certificate that must be present in the plea. This certificate attests that the shuttle program was installed by the shuttle company. This permits the client to distinguish a real shuttle from someone pretending to be one by installing the same program. The key used in this certificate is the public key of the shuttle company: this is loaded on the principal when the customer program is installed.

4.3 Analysis of Model

Message quality is useful in addressing the risks outlined in Section 2. With respect to theft, while this risk can never be eliminated, the goal of a security framework is to reduce the benefit to a thief. For instance, employing PINs to access a device and having the owners re-enter his PIN at the start of each session, reduces benefit to thieves. For this reason, PINs are part of our security solution. Further, the plausibility feature of message quality ensures that if Charlie steals Alice's device and sends a message to Bob, then Bob can detect that the device is stolen if the request does not correspond to a message that Alice would normally send. The thief can only use the device for actions (behavior) that Alice specified, and only in the time window that exists before the principal owner is required to re-enter a PIN.

Similarly, viruses exhibit their presence on a device through behavior that is not typical of the owner of the device. Malware that manifests itself in this way is detected through violations of plausibility since messages are sent that are incompatible with the device's profile. Thus, even if validly installed programs get corrupted via stack smashing or buffer overflow, their invalid behavior gets detected. Finally, the usefulness and trustworthiness properties tackle spam.

Security is a feature that is inherently difficult to measure. First, a system is only secure with respect to some policy. In ambient systems, each principal has his own policy on the messages that are acceptable. There need be no relation between the policies of different principals. The measure of security for a principal is the ratio of correct message acceptance and rejection decisions made by the automated security framework compared to the outcome that the principal would attribute if he were taking the decision himself, manually, for each message [2]. In this sense, security is measurable using a Turing-like test.

To consider the security of the model, we can compare it with other security approaches. There are traditionally two phases in a security infrastructure: authentication and

authorization. Authentication is concerned with binding a principal to a virtual identity representation within the system; authorization is concerned with determining whether the principal associated with the virtual identity may have access to a service or information item. Every automated security infrastructure works with two assumptions:

- Assumption U: Any principal authenticated and bound to a virtual identity for p is p.
- Assumption T: Any principal p that has been granted access to a service is trusted to use that service.

All attacks on security infrastructures seek to undermine these assumptions. For instance, key theft or password cracking undermine Assumption U; behaving well in order to become trusted so as to exploit this trust to misuse information undermines Assumption T. Security infrastructures integrate mechanisms to enforce the assumptions. For instance, certificate revocation limits curtail attacks on Assumption U; audits seek to curtail attacks on Assumption T.

Practical and theoretical limits means that the two assumptions can never be satisfied. For Assumption T, the theoretical limit is that there can be no guarantee that a principal that has always behaved correctly in the past will continue to behave correctly in the future. He may become dishonest or his device become faulty through ware or virus infection. For Assumption U, a user may be acting under the orders of an attacker (through blackmail or threats), so the effective user is the attacker.

Nonetheless, the assumptions do allow us to reason about the security of our model compared to other security infrastructures. For Assumption U, our model defines the window of attack as being the session during which a PIN is not required, and the set of programs that the owner has installed on the device. Thus, the device can be temporarily used by a thief for the same purpose that the owner uses it. Nonetheless, the thief cannot install his own programs unless he has successfully carried out a PIN theft. This is a stronger guarantee than currently exists for PC environments.

With respect to assumption T, the innovation of trust and recommendation frameworks with respect to traditional architectures is that the dynamic behavior of principals is taken into account. In other words, the privileges accorded to a principal are directly related to its past behavior; a principal can increase its trustworthiness by behaving correctly. This exploits Assumption T, i.e., the more one knows about a principal's past behavior, the better one can judge its future behavior. In traditional architectures, the system is not able to automatically adapt the privileges accorded to a principal as a function of its behavior. Our system allows dynamic behavior to be judged since evidence must be transfered as part of the plea. Evidence cannot be faked by a principal since it is automatically registered in the trusted zone during communication. Thus, the only way to obtain evidence of having called a taxi is to call a taxi.

In the model, the history is stored in the trusted zone. A principal may choose to remove parts of its history from the trusted zone – for instance to economize space or to eliminate redundancy – but it may not add messages explicitly. We contend that most scenarios only need to record a small part of their history. The shuttle service was an example where a

principal's history is used as evidence to argue for a message's quality. Another approach to eliminating the history log is for a principal to ask a well-known authority principal to sign a profile certificate (with a role historyValidatedBy) for the requesting principal. The requestor only needs to present its history log in exchange for the profile certificate. The principal can transmit this profile certificate in pleas.

5 Related Work and Conclusions

This paper has presented a security model for ambient information systems. The model concentrates on the property of message quality, which is the cornerstone for implementing other security protocols. The framework has the advantage of not undermining the attractive properties of ambient systems, notably, anonymity and spontaneity of communication. The model is prototyped in Java and the prototype's implementation uses the TPM. A longer version of this paper is found in [1].

Property-based attestation [15] looks at how the TPM can be used to enable devices to deliver proofs that specific security properties hold. The work does not specify how properties are derived from the measures taken by the TPM. Nonetheless, it shows that the TPM can be used for more elaborate security guarantees than binary (code) attestation.

The plausibility aspect of message quality is similar in concept to proof-carrying code [12] whose main aim is to protect a platform from untrustworthy code. In this approach, a downloaded program is accompanied by a proof of the program's (good) behavior. The host environment can verify the proof mechanically, and if this passes, can then trust the program to run securely. An example of the properties that can be proven in this approach is the sequence of system calls made by the program, e.g., [17] where the host environment verifies that the program does not leak platform information. The message quality model is nonetheless computationally less expensive than the verification mechanisms of proof-carrying code.

The goal of reputation management systems [18] and trust systems is to enable principals to exchange their experiences with other principals so that one can judge their expected behavior. However, two key challenges remain in reputation management. First, there is no way of generalizing the behavior of principals across applications. For instance, the fact that a principal can be trusted to forward packets in an *ad hoc* network, does not mean that one can trust it to store a file. Second, in a real environment, a principal might have a limited number of other devices that it can contact at a given moment for recommendations. This is known as the sparseness problem; it underlines that fact that reputation systems require that a principal can access a large amount of reputation data for it to be able to take a meaningful decision. These issues are less important in this paper's model since it attests program behavior directly. Nonetheless, we leverage aspects of trust based systems by allowing principals to store a history of their past behavior (message exchanges). This evidence constitutes trust data that a principal can use to convince another that it is trustworthy.

The notion of message quality is important to a wide class of systems today where information is pushed to the user, e.g., e-mail, instant messaging, RSS feeds, etc. Even when

surfing the Web, users are inundated with information, so mechanisms for verifying the quality of this information are required. Whereas localized networks in ambient computing reduce the dependence of devices on trusted third parties, scalability and performance reasons can force this dependence in Web based systems to be reduced.

Acknowledgments This work is partly conducted in the context of the PRIAM (Privacy in Ambient Systems) project - an INRIA financed collaboration examining the representation of privacy legislation in modern information systems.

References

[1] C. Bryce. Message quality for ambient system security. Technical Report P11896, IRISA, 2008.

[2] C. Bryce, N. Dimmock, K. Krukow, J.-M. Seigneur, V. Cahill, and W. Wagealla. Towards an evaluation methodology for computational trust systems. In P. Herrmann, V. Issarny, and S. Shiu, editors, *iTrust*, volume 3477 of *Lecture Notes in Computer Science*, pages 289–304. Springer, 2005.

[3] C. Bryce, C. Razafimahefa, and M. Pawlak. Lana: An approach to programming autonomous systems. *Lecture Notes in Computer Science*, 2374:281–298, 2002.

[4] P. Couderc and M. Banâtre. Ambient computing applications: an experience with the SPREAD approach. In *HICSS*, page 291, 2003.

[5] Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems (IPTP-S), LNCS*, volume 1, 2002.

[6] FBI/CSI. 12th annual csi/fbi computer crime and security survey, 2007.

[7] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, Jan. 1985.

[8] J. Gosling, B. Joy, G. Steele, and G. Bracha. *The Java Language Specification, Third Edition*. The Java Series. Addison-Wesley, Boston, Mass., 2005.

[9] D. Konstantas, V. Jones, and R. Herzog. Mobihealth - innovative 2.5/3G mobile services and applications for health care". In *IST Mobile & wireless telecommunications Summit 2002*. Thessaloniki, Greece, 17-19 June 2002.

[10] D. Mckitterick and J. Dowling. State of the art review of mobile payment technology. Technical report, June 13 2003.

[11] MetaGroup. Spam, viruses, and content compliance: An opportunity to strategically respond to immediate tactical concerns. Technical Report 800-945-META [6382], International Computer Science Institute, Jan. 2005.

[12] G. C. Necula. Proof-carrying code. In *POPL*, pages 106–119, 1997.

[13] P. Pellegrino, D. Bonino, and F. Corno. Domotic house gateway. In H. Haddad, editor, *SAC*, pages 1915–1920. ACM, 2006.

[14] G. P. Picco, A. L. Murphy, and G.-C. Roman. LIME: Linda meets mobility. In *Proceedings of the 21st International Conference on Software Engineering*, pages 368–377. ACM Press, May 1999.

[15] J. Poritz, M. Schunter, E. Van Herreweghen, and M. Waidner. Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical Report RZ 3548, IBM Research, May 2004.

[16] R. Rivest, A. Shamir, and L. Adleman. On digital signatures and public key cryptosystems. *Comm. A.C.M.*, 21:120–126, 1978.

[17] R. Sekar, V. N. Venkatakrishnan, S. Basu, S. Bhatkar, and D. C. DuVarney. Model-carrying code: a practical approach for safe execution of untrusted applications. In *SOSP*, pages 15–28, 2003.

[18] V. Shmatikov and C. L. Talcott. Reputation-based trust management. *Journal of Computer Security*, 13(1):167–190, 2005.

[19] Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 85, Trusted Computing Group, Feb. 2005.

[20] N. Zannone. A survey on trust management languages [reduced]. Technical report, Aug. 01 2004.