Message Quality for Pervasive System Security

Ciarán Bryce

Abstract. Security is a challenge in pervasive systems for several reasons. First, the large number of peers challenges the use of centralised security services like certificate authorities and reputation services. Second, a principal may be a physical object whose identity might not convey useful information for taking security decisions. Establishing a trusted channel depends more on a principal being able to demonstrate what is does, rather than who it is. Third, device mobility leads to the formation of *ad hoc* networks. Devices in these networks may be interacting for the first time so devices must carry sufficient information for them to establish mutual trust with other devices. This paper proposes a security model for pervasive systems, based on the idea of message quality. The model allows a principal to establish the intent of an adversary and to make the adversary prove its trustworthiness by furnishing proof of past behavior.

1 Introduction

The technological combination of portable devices (e.g., smart phones, PDAs), wireless networking and processor cards embedded in everyday devices has led to the emergence of pervasive computing systems. These systems support a rich set of person-centric applications, like electronic payment and mobile health [11], and are also used in process control systems. Security is a major concern for pervasive systems, especially with personal devices containing private information and increasingly sensitive applications.

Pervasive systems pose a challenge for information security. One reason is that there can be millions of devices, and coupled with mobility and wireless communication, devices can communicate while in *ad hoc* networks. Communicating devices can be unknown to each other, and the network might not possess a trusted third party that can act as a certificate authority [20] or reputation server [23] and thus facilitate the establishment of trusted channels between devices. Solutions for security need to be scalable. This means that when two peers want to establish a trusted channel, then there should be sufficient information on their devices to establish the channel, without having to rely on third parties.

Before reflecting on mechanisms for pervasive system security, it is worth reflecting on the purpose of security in these systems. The major security risks that are considered in this paper are PDA theft or loss, virus infections, device ware, as well as SPAM where a device is sent useless or untruthful messages. This paper presents a security model and implementation that seeks to address these concerns.

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 a pervasive system security 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.

The model is centered on the notion of *message quality* for each message M sent from Alice to Bob. The idea is that when Bob receives m, he needs to be decide whether he can act on m based on his trust in Alice. Message quality is defined as the following general properties; more precision will be given later in the paper.

- *Fidelity*: the message M is typical of a message that Alice utters. In other words, the message is consistent with the expected behavior of Alice. As we will see, this property is useful for detecting stolen devices and viruses.
- *Freshness*: M originates from Alice, and is not being replayed by her. This property is useful for detecting spoofing attacks.
- *Trustworthy*: M is probably true, that is, the claim contained in M can be supported by evidence that Alice provides. This property leverages the approach of trust frameworks.
- *Utility*: M is interesting to Bob, as opposed to a SPAM message that wastes his time and resources.

The remainder of the paper elaborates on the notion of message quality. The paper also presents a prototype implementation that relies on the Trusted Platform Module (TPM) [24]. The TPM is used to build a trusted zone on the device that stores a profile of running programs as well as a history of messages exchanged by the device. The profile and history are sent along with each message M and enable Bob to verify the quality of M.

This paper is organized as follows. Section 2 presents the security challenges that the paper addresses. The notion of message quality is defined in Section 3, and a model is formalized along with an implementation strategy. In Section 4, the model is integrated into a programming model that is often used in pervasive computing – the tuple space model [8]. Finally, an example of the security model in use is presented and an implementation is described. Related work is presented in Section 5 and Section 6 concludes.

2 The Pervasive System Security Challenge

The term *pervasive computing* was coined by Weiser [25]. As suggested in Figure 2, such a system is made up of heterogeneous devices. Some may be human operated, such as a mobile phone or Personal Digital Assistant (PDA); others are processor

and wireless network cards embedded in everyday appliances. Estimates suggest that more than 98% of computing devices are embedded [6]. Communication between peers can be explicit – as in Figure 2 where one peer sends out a search message in the network - or implicit where the peers detect each other's presence and react accordingly, e.g., a door opens when a home owner approaches whose RFID badge gets detected. The enabling technologies of these networks include PDAs and smartphones, as well as wireless communication technologies, e.g., Bluetooth, WLAN, GPRS, etc. Pervasive systems are employed for process control systems and person-centric applications, e.g., mobile health [11], domotics [17] and payment [13]. A characteristic of all of these systems is that communication is generally localized to devices in close physical proximity.

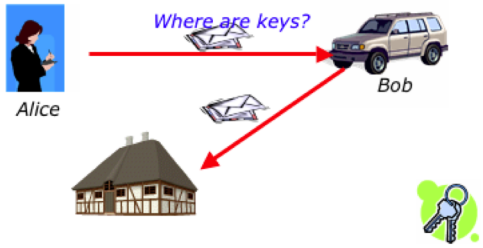

Fig. 1. Pervasive System

According to the latest FBI/CSI report on information security [7], viruses are now the most common and most costly source of security attacks. 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). Further, device mobility can help a virus pass behind corporate firewalls without having to attack them.

A second major security concern in today's information systems is information misuse [7]. This is the problem of information being improperly exposed or destroyed through inefficient access controls to system information and resources. A concrete example of this 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 pervasive systems is SPAM – a well known problem for the Internet, with up to 70% of e-mail traffic consisting of unsolicited messages [14]. SPAM is attractive for attackers (SPAMers) due to the low cost of sending messages.

[1] http://www.virusthreatcenter.com/

[2] Association Francaise des Opérateurs Mobiles (AFOM); http://www.afom.fr/v3/TEMPLATES/acces_elus_l2.php?rubrique_ID=115

A similar situation can arise in pervasive 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 messages from items on sale.

A pervasive system security model must address these risks. However, there are challenges implementing security. The first is the potentially huge number of peers without centralized management. No peer knows all others, and most peers know few others. Strictly speaking, a 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 connectivity of wireless networks, these 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 contains sufficient data to ensure that the message is securely transferred and for Bob to verify that it is safe to act upon its contents.

Another security challenge that also stems from decentralization relates to message authentication. In traditional security protocols, authentication allows one to establish that a message is fresh, and is not being replayed by an attacker. Failure to do so makes the system vulnerable to man-in-the-middle or spoofing attacks – this happens when an attacker Charlie manages to make Alice believe that he is Bob, and makes Bob believe that he is Alice. This impersonation allows Charlie to subvert the conversation between Alice and Bob. Spoofing attacks are a high risk in pervasive systems since, as mentioned, the identity of a peer is difficult to establish. In addition, an autonomous entity may be able to generate (false) identities – this is the Sybil attack [5].

These requirements suggest that a shift is required in how we reason about security. This paper proposes a security model that addresses these concerns and that can be integrated into a programming model for pervasive applications. Not all security concerns are addressed by this model. For instance, it does not ensure the integrity of the network, i.e., protect transceivers from interference with the network signal, or prevent denial of service attacks where devices are inundated with messages.

3 Message Quality and Security

This section presents a security model designed to permit a principal (device) to take a security decision about a message it receives. The decision to take is to either accept or reject the message.

The core notion of the model is *message quality* – the property that when Alice sends a message to Bob, Alice can demonstrate her expected behavior and argue that she has behaved in a trustworthy manner in the past. Doing so is useful for security since Bob can estimate the trustworthiness of the message and verify that the message is fresh, rather than a message being replayed. Further, Bob can believe that Alice's device is not virus infected, runs software that was legitimately installed, and that Alice is likely not using a stolen device.

Section 3.1 presents the adversarial model; Section 3.2 introduces the idea of message quality, and argues that it addresses the security requirements of the preceding section. Section 3.3 formalizes message quality, and looks at the notion of principal in more depth.

3.1 Adversarial Model

A principal Alice coexists with a large sized community of principals in ambient systems. There might be no centralized control and no globally trusted authority that can mediate in the establishment of trusted channels between principals. Further, due to the use of wireless communication links, the network neighborhood of a principal can vary greatly over time, and a principal is not guaranteed to be able to contact a trusted server (e.g., certificate authority or reputation server) whenever it needs to take a security decision.

The security decision that Alice needs to take is, for each message m received, whether the message should be accepted or rejected. Each m impacts on the behavior of a principal and trust in m is the basis of establishing confidence in the sending principal – this being required for information sharing and service access.

Each principal owner is autonomous and has control over his device. An owner can install any software on his device, access any service or manipulate information on the device in any way. A device may be stolen and then used or misused by a non-owner.

We assume that each principal's device contains a *trusted zone* whose integrity and correct functioning can be verified by partner principals. The information in the trusted zone can only be set using a PIN protocol (which presumedly a thief cannot know). Even if an owner manages to corrupts its trusted zone, a partner principal can detect this integrity loss. As suggested by Figure 3.1, the mechanisms required to implement trusted zone integrity are furnished by the Trusted Computing Group's architecture [24].

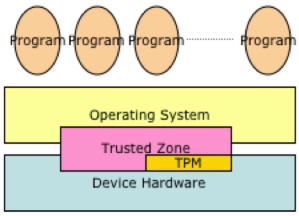

Fig. 2. Pervasive device environment

As will be seen, the trusted zone is used to store a profile of the principal's behavior and a history of the principal's past behavior. The current behavior is defined by the currently installed programs, the past behavior by the messages previously exchanged.

3.2 Message Quality

Consider a message M sent from Alice to Bob, c.f., Figure 3.2. There are four properties of this exchange that define the quality of M.

- **Fidelity**. 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 bus passenger, then the messages she is likely to utter include requests for a bus.
- **Freshness**. This property states that M originates from Alice, and is not a message that she is replaying from a third party. For instance, if Alice says that messages addressed to her should be encrypted with k, then Bob can believe that the message is not from Charlie who is trying to masquerade as Alice.
- **Trustworthiness**. 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 bus, then Bob – the bus driver – can stop the bus with sufficient confidence that there is a passenger waiting to mount. Trustworthiness is a stronger property than fidelity. It 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., [23, 26].
- **Utility**. 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.

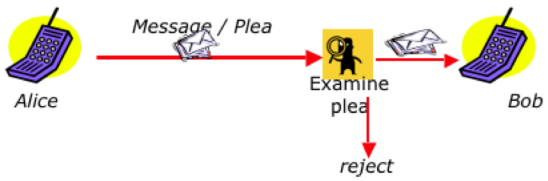

Fig. 3. Message Quality

Message quality relies on a principal being able to prove its expected current behavior (the profile stored in its trusted zone) and to demonstrate proof of good behavior in the past. We contend that this is sufficient for the four properties listed above, and that message quality is a way of addressing information security in pervasive systems. For instance, 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 fidelity since messages are sent that are incompatible with the device's profile. Similarly, fidelity 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.

For further strength against theft, the configuration could oblige the owner to play a PIN protocol each time an application gets launched.

3.3 Model

There are two aspects to the model: the way that principals are identified and the elements of a message needed to verify quality. We look at each in turn, before formalizing the properties of the model.

Principal Attestation

The term *principal* has been used in computer security for many years to denote an entity whose actions need to be controlled [21]. In distributed computing, the actions of servers and machines need to be controlled so these are also considered as principals. Principal authentication uses public-key cryptography: a principal proves its identity by signing data with its private key, and the signature can be validated with the corresponding public key. A certificate authority principal issues certificates that attest to the binding of a principal's name to a public key. Today, the term principal is defined as an entity that can be denoted by a public key [12].

In many security infrastructures, a public key is seen as a principal identity, though this may be aliased with a more human readable name (e.g., "John Smith"). Security is generally enforced in two stages: authentication verifies the binding of a principal to a public key (identity) and authorization determines the legality of each action according to the established identity. The notion of identity plays a central role but this situation needs to change in security infrastructures for pervasive systems, for several reasons.

- Scalability problems arise due to the potentially huge number of principals. When two communicating principals do not recognize each other's certificate authority (CA), then a chain of CAs needs to be established from the authenticating principal to the principal being authenticated. Each CA attests to the validity of the public key of the next CA in the chain. These chains can become lengthy in a large population and this penalizes authentication. Further, in pervasive systems, alias management (for meaningful principal names) is also an overhead, since many principals are ordinary devices, e.g., cars, doors, chairs, etc., that require more manageable names than simple serial numbers.
- 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). Thus, a hitherto trusted principal can start behaving badly, so a security infrastructure must be able to detect this.

As already mentioned, it is more important in pervasive systems for a principal to prove *what it does* rather than *who it is* (i.e., its identity). A principal's identity does not change, but its ability to service a request – what it does – can change as new functionality is added, viruses spread, etc. It is important to be able to detect

these changes. For instance, it is more useful for a vending machine to prove that it returns a soda in return for a coin than it is for it to prove its serial number. It must be able to show that previous interactions were successful, i.e., that nobody gave a coin without a soda being returned.

In the message quality paradigm, a principal is defined as an active entity that may undertake actions, and with which a definable behavior may be associated. The formal definition of behavior is given shortly. Thus, when we say Alice sends a request to Bob, we mean that an entity with the behavioral profile denoted by Alice sends a request to a principal whose behavior is denoted by Bob.

Obviously, some applications require identity-based controls, e.g., the door only opens when an employee is detected. Typically, this is implemented by having the employee device play a standard authentication protocol with the door, in which he proves possession of a secret (key), e.g., [16]. However, this protocol can also be expressed in the message quality model. For instance, when the employee device transmits the employee ID value or the signed nonce, the message quality model verifies that the employee device is not running a rogue program that allows it to masquerade as the employee – this check relies on message quality. An alternative approach is to have the signed nonce specified in the employee profile, thus integrating authentication directly in the message quality model.

Principal Structure

The structure of a pervasive device platform for our model is illustrated in Figure 3.1. The trusted zone stores the principal's behavior, or *profile*, and is protected from tampering by user programs. The profile may be set using a PIN, which assumedly, only the owner of the device should know. The model does not specify how the trusted zone is implemented. Our own implementation uses a Java environment to help ensure that programs do not gain access to trusted zone state, and relies on (a software emulation of) the TPM [24] to prove to partners that the trusted zone has not been tampered with.

Profiles

As mentioned, principal behavior is defined in a profile. More precisely, 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. This can be expressed as the set of messages that the program sends and receives.
- A certification of the program origin that describes where or by whom the program was developed. (The createdBy certificate).
- A certification of the program installation describing who installed the program on the device. (The installedBy certificate).
- Any other application or service specific certifications that are considered useful in an application context, e.g., inspectedBy.

The certificates in the profile are termed *profile certificates*.

The role of a certificate is to bind a public key to a profile, which means that the model does rely on well-known principals. We conjecture that the number of certificates is nonetheless minimized since they certify behavior, and not identity. These certificates can be downloaded with the software at installation time. The well-known principals are the software or service providers, which become known when the user subscribes to their service by installing client software (e.g., taxi client) on his device.

Pleas

Recall that when Alice sends a message to Bob, the onus is on Alice to demonstrate the quality of the message to Bob. To that end, she constructs a *plea* object that argues for her message, c.f., Figure 3.2.

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. The history is also stored in the trusted zone of the principal. The 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 to the history. We contend that most scenarios only need to record a small part of their history. Section 4 presents an example where a principal's history is used as evidence to argue for a message's quality.

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.

- Fidelity is verified by i): ensuring the M belongs to Alice's profile; ii) validating any profile certificates in the profile.
- Trustworthiness is verified by verifying that the history of messages in Alice's history match a series of message exchanges that Bob specifies as required evidence.
- Utility is verified by ensuring that M belongs to Bob's profile.
- Freshness is verified by ensuring that M was created on Alice's device. The trusted zone is required to enforce this property.

Model Properties

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.

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

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

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

A message m exchanged between two devices is a quadruple:

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

Message quality is defined as freshness, fidelity, utility and trustworthy. Formally, freshness and fidelity are specified in the same way, since in both cases the binding of a message to a profile is verified. We thus enforce the three following properties for fidelity/freshness (σ_F) with respect to a role r and key k, utility (σ_U) with respect to the receiver's profile p', and trustworthiness with respect to a required history (evidence) h' (σ_T).

i) $\sigma_F[r, k](m) \triangleq m.d \sqsubseteq m.p \ \wedge \ m.c(r)(p) = k$

ii) $\sigma_T[h'](m) \triangleq h' \sqsubseteq m.h$

iii) $\sigma_U[p'](m) \triangleq m.d \sqsubseteq p'$

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:

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

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.

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

4 Programming Model and Example

This section describes an implementation of message quality in a tuple space programming model. The section opens with a description of the programming model. A Java API is presented in 4.2 and an example is presented in 4.3.

4.1 Programming Model

Our goal is to implement message quality as transparently as possible into a programming model for pervasive applications. A transparent implementation of message quality is one that entails little or no modification to the programming model primitives.

The programming model chosen in this paper is based on the Linda tuple space model [8]. 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 `<"Airport", String, int>`. The Linda operation to read this tuple from the tuple space is read(`<"Airport", String, int>`). The operation blocks if there is no matching tuple present, and only unblocks when a matching tuple is written to the tuple space by another principal.

The advantage of the tuple space model is that it is *anonymous* and *connectionless*. The model is anonymous in that 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 in pervasive systems where there might be no principal naming scheme in place. The tuple space model is connectionless in the sense that a sender and receiver need not be present at the same time, unlike for socket communication. The connectionless property is particularly useful in pervasive systems where networks of principals are *ad hoc*, meaning that a sender can disappear from the network before its message gets read.

Our implementation is based on the Lana system [2], 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 4.1. Many systems designed for pervasive environments employ Linda's tuple space model for the reasons cited above, e.g., Lime [18], Spread [4].

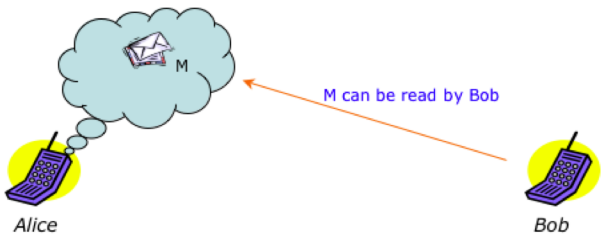

Fig. 4. Each device has its own tuple space

The tuple space operations of our model are given in Figure 4.1. The read operation returns a tuple matching the pattern argument from the tuple space of any device in the neighborhood of the device issuing the request. The operation is non-blocking, so if no device is present in the network, or if there is no matching tuple, the operation returns an empty tuple. The reading device 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 and remove erases it from the space. Note that the tuple space primitives do not contain explicit reference to the message quality model, so we can consider its implementation in the programming model to be transparent.

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

Fig. 5. Primitives of a tuple space model using message quality

Message quality is a property relating to a message sent from Alice to Bob. In our implementation, a message M sent from Alice to Bob encapsulates a tuple that is in fact a reply to a preceding read request. This is illustrated in Figure 4.1. The request from Bob includes TPM information such as the index of the PCR registers to use[3]. These values are generated by the trusted zone integrated in the tuple space runtime and are transparent at the application programming level.

4.2 Prototype

We implemented the tuple space framework in the Java programming language, this choice being facilitated by the widespread availability of Java environments for pervasive 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. An extract of key classes and of some of their methods is given in Figure 7. 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.

[3] PCRs store hashes of the code running on the device. A partner challenges the device to produce copies of some hashes which it uses to compare to expected hash values and thus see if the software has not been tampered with.

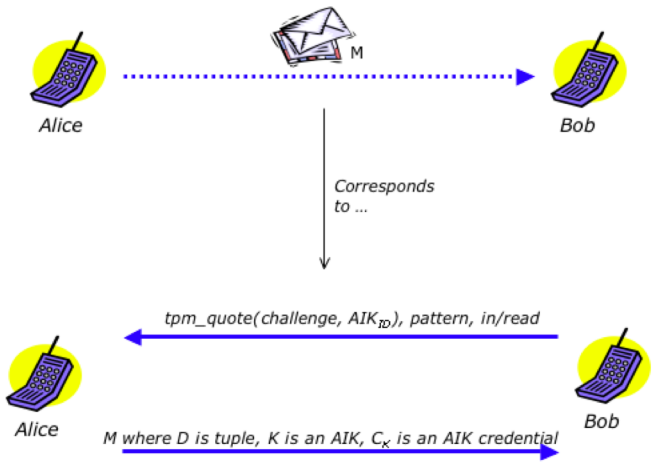

Fig. 6. Implementation of message quality in tuple space model.

The classes Tuple, TupleSpace, and tuple Entry implement the core tuple space abstractions. The TupleSequence class describes a tuple exchange; this is used by the trusted zone to represent the list of actions of a program.

TrustedZone is a public class that implements all trusted zone functionality. This class is used by clients for key operations on the trusted zone, e.g., to set a new profile (setProfile), to specify the evidence required for an incoming message to be validated (setRequiredEvidence), as well as to specify the profile certificates required when validating message quality (addRequiredCertificate). TrustedZone can also be used to view the platform's message history; the history may be truncated in order to economize memory space. This is achieved using the viewHistory and selectHistory operations. Our trusted zone contains a software emulation of the TPM.

A principal is represented by the Device class. The local Device can be used to query the network for other devices. This operation is used internally by the runtime for the read operation to select devices to query for tuples. Our Device class contains two implementations of network query: one for a local area IP network where a multicast address is used to implement the query, another for a Bluetooth network which is built over the BlueZ stack.

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 [9] and C-sharp [10]. Another approach is to rely on hardware protection: this is now feasible thanks to the Trusted Platform Module (TPM) – a hardware device whose functionality is specified by the Trusted Computing Group [24]. The TPM is becoming commodity hardware, with 100 million TPM-enabled PCs having now been shipped, and Windows Vista relying on the TPM for its security model.

```
public final class TupleSpace {
	public static Tuple rd(Tuple pattern);
	public static void out(Tuple tuple);
	public static void remove(Tuple tuple);
}

public final class Tuple {
	public Tuple(TupleEntry[] entries);
	public Entry get(int index);
	public boolean match(Tuple pattern);
}

public final class Entry {
	public static class Entry.Any; // Wildcard
	public static class Entry.Int;
	public static class Entry.String;
	public static class Entry.Type; // For pattern entry, when searching for object of some type
	public boolean match(Entry pattern);
}

public final class TupleSequence {
	public static interface ExchangedTuple;
	public static class InTuple; // Implements ExchangedTuple
	public static class OutTuple; // Implements ExchangedTuple
	public TupleSequence(ExchangedTuple[] tuples);
	public boolean matches(TupleSequence me);
	public boolean contains(TupleSequence me); // This implements the ⊑ operator
}

public final class TrustedZone {
	public void registerProfileCertificate(PIN, ProfileCertificate);
	public void setRequiredEvidence(TupleSequence evidence);
	public void addRequiredCertificate(PIN, ProfileCertificate);
	public void setProfile(PIN pin, TupleSequence profile);
	public TupleSequence viewHistory();
	public TupleSequence selectHistory(int index1, int index2);
	// Some internal functionalities
	void validate(Message message, int pcrIndex, int nonce);
	Message makePlea(int aikIndex, int pcrIndex, int nonce, Tuple t);
}

public final class Device {
	private class TrustedZone;
	public Device[] getNeighbouringDevices();
}
```

Fig. 7. Extract of Java API.

The TPM is a trusted unit whose role is to attest to the integrity of software running on the device – the trusted zone and runtime environment in our case. A TPM takes measures of the software in the form of hashes and stores these in internal registers known as PCRs. These measures are verified with respect to validation certificates that are issued (by the runtime environment provider) when the environment is installed on a device.

When Alice sends her message M in a message, her trusted zone constructs the following data sequence from M:

$$\{ M, K, S, C_K \}$$

K is a public key that is generated by the TPM of Alice (using the TPM's createWrapKey operation) and C_K is a TPM generated certificate for K (using the TPM's certifyKey operation). K is signed using a TPM key (AIK in TPM parlance) that is distributed to principals with the runtime environment installation. S is a signature for the message; its format is $\{ M, pcr^* \}$ – the message and a series of PCR entries – hashed and signed with K.

4.3 Example

This section illustrates an application where message quality is used. Some details are simplified for the purposes of the paper.

The example illustrates a shuttle service that a 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 4.3. 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 the "Bye" message.

There are two key security requirements that we want to implement in this application. One is fidelity for shuttles – customers can be sure that they are communicating with real shuttles, rather than rogue devices masquerading as shuttles. The second requirement is customer trustworthiness: shuttles that received 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. 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 price paid by customers is that their first ride via the service is on a slow shuttle.

The first code extract 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 for the part of taking a customer. Recall that the principal is unable to send messages

Fig. 8. Shuttle Scenario

that do not correspond to its profile since they 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. However, its first task is to define the evidence required 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);
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 real buses from people pretending to be one by installing the same program. The key used in this certificate is the public key of the shuttle company: this can be loaded onto the principal when the customer program is installed.

5 Related Work

There has been much work on *reputation management systems* [23] and trust systems. Their goal 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.

Security in sensor networks is a further domain of interest to many researchers. Many traditional security problems exist, including eavesdropping, tampering, traffic

analysis and denial of service. However, most sensor networks have a centralised network structure, so security is currently oriented to the implementation of secure group protocols [3]. However, as the number of devices becomes extremely large, decentralised solutions will have to be prioritised.

Property-based attestation [19] 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 security guarantees that are more elaborate than binary attestation.

Message quality is similar in concept to proof-carrying code [15] 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., [22] where the host environment verifies that the program does not leak platform information. Related to this is work on PolicyMaker [1], where a security model for distributed systems is presented that deprecates identity: certificates are used to attest to the right of the holder to execute an action, irrespective of its identity. However, this model does not say how the certification comes about, and does not treat the issue of changing trust in entities over time.

6 Conclusions

This paper has presented a security model for pervasive 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 pervasive systems, notably, anonymity and spontaneity of communication. The model is prototyped in a Java environment and the prototype's implementation uses the Trusted Platform Module.

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 pervasive computing reduce the dependence of devices on trusted third parties, scalability and performance reasons can force dependence in Web based systems to be reduced.

References

1. Blaze, Feigenbaum, and Lacy. Decentralized trust management. In *RSP: 17th IEEE Computer Society Symposium on Research in Security and Privacy*, 1996.

2. Ciarán Bryce, Chrislain Razafimahefa, and Michel Pawlak. Lana: An approach to programming autonomous systems. *Lecture Notes in Computer Science*, 2374:281–298, 2002.
3. Haowen Chan and Adrian Perrig. Security and privacy in sensor networks. *IEEE Computer*, 36(10):103–105, 2003.
4. Paul Couderc and Michel 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 (IPTPS), LNCS*, volume 1, 2002.
6. Deborah Estrin, Ramesh Govindan, and John S. Heidemann. Embedding the internet: Introduction. *Commun. ACM*, 43(5):38–41, 2000.
7. FBI/CSI. 10th annual csi/fbi computer crime and security survey, 2005.
8. David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.
9. James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification, Third Edition*. The Java Series. Addison-Wesley, Boston, Mass., 2005.
10. E. Gunnerson. *A Programmer's Introduction to C#.* Apress, 2001.
11. Dimitri Konstantas, Val Jones, and Rainer 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.
12. Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, November 1992.
13. David Mckitterick and Jim Dowling. State of the art review of mobile payment technology. Technical report, June 13 2003.
14. 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, January 2005.
15. George C. Necula. Proof-carrying code. In *POPL*, pages 106–119, 1997.
16. R. Needham and M. Schroeder. Authentication revisited. *ACM Operating Systems Review*, 21(1):7, January 1987.
17. Paolo Pellegrino, Dario Bonino, and Fulvio Corno. Domotic house gateway. In Hisham Haddad, editor, *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), Dijon, France, April 23-27, 2006*, pages 1915–1920. ACM, 2006.
18. Gian Pietro Picco, Amy L. Murphy, and Gruia-Catalin Roman. LIME: Linda meets mobility. In *Proceedings of the 21st International Conference on Software Engineering*, pages 368–377. ACM Press, May 1999.
19. Jonathan Poritz, Matthias Schunter, Els Van Herreweghen, and Michael Waidner. Property attestation—scalable and privacy-friendly security assessment of peer computers. Technical Report RZ 3548, IBM Research, May 2004.
20. R. Rivest, A. Shamir, and L. Adleman. On digital signatures and public key cryptosystems. *Comm. A.C.M.*, 21:120–126, 1978.
21. Jerome H. Saltzer. Protection and the control of information sharing in Multics. *j-CACM*, 17(7):388–402, July 1974.
22. R. Sekar, V. N. Venkatakrishnan, Samik Basu, Sandeep Bhatkar, and Daniel C. DuVarney. Model-carrying code: a practical approach for safe execution of untrusted applications. In *SOSP*, pages 15–28, 2003.
23. Vitaly Shmatikov and Carolyn L. Talcott. Reputation-based trust management. *Journal of Computer Security*, 13(1):167–190, 2005.

24. Trusted Computing Group. TPM main specification. Main Specification Version 1.2 rev. 85, Trusted Computing Group, February 2005.
25. Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.
26. Nicola Zannone. A survey on trust management languages [reduced]. Technical report, August 01 2004.