Peer-to-Peer Single Sign-On Security Scheme for Today's Smart Spaces*

Enrique Soriano Salvador
Francisco J. Ballesteros
Gorka Guardiola

Laboratorio de Sistemas
DITTE, Universidad Rey Juan Carlos de Madrid
{esoriano,nemo,paurea}@lsub.org

Abstract. Nowadays, computing environments are almost ubiquitous systems. Users own several machines and need to manage them in a non-obtrusive and transparent way. In addition, users eventually need to share their devices with other people. Traditional Single Sign-On (SSO) schemes do not handle well scenarios where users work simultaneously with more than one device because their agents do not cooperate. Besides, they usually depend on centralized services and are hard to administer. In this work we present SHAD, a decentralized human centered and Peer-to-Peer security agent that besides offering SSO allows users to share their devices in a natural way.

1 Introduction

Ubiquitous computing is becoming a reality. Nowadays, research labs and offices are getting closer to the scenarios described by Mark Weiser in "The Computer for the 21st Century"[41]. Hardware has become cheap enough and users own multiple heterogeneous devices. Therefore, users need to work with their devices simultaneously and share them in a non-obtrusive way.

Traditional Single Sign-On schemes work well when the principal is using a single workstation and accessing classic client/server services through network connections. The problem arises when the user needs to physically use several devices at same time.

Principals own several devices to perform different tasks on each one. As a consequence, users have to log on and authenticate for each device. Traditional SSO systems do not work well in this case, because the user has to type at least one password per machine. Therefore, **there is not a single sign-on**.

Several authentication devices have been proposed in order to reduce the level of obtrusiveness, see for example biometric sensors [24], Smartcards [31], Ibuttons [1] or USB Tokens [4]. They alleviate the problem, but they are still obtrusive. Most of them depend on specific hardware (generally readers) that

* Research work supported by the Spanish Ministry of Science and Technology project TIN2004-07474-C02-02.

may not be suitable to be used in some situations, specially with mobile devices. A user cannot be inserting cards in readers or pressing her fingerprint onto biometric devices all the time.

We argue that explicit authentication is an obstacle to vanish the machines and make the environment become pervasive, and propose a mechanism to overcome this obstacle.

Sharing devices in an ubiquitous environment is also a problem. As stated by Stajano [39], in this kind of systems some tasks are the result of the synergy between many devices. Usually, these devices do not belong to the same user. As a consequence, we need news security schemes for sharing people's resources.

Network Information Service[5], Kerberos[40], Sesame[27] and related systems are based on centralized servers that authenticate users and manage the access control policies. This makes the inclusion of new elements in the system hard and non-immediate. When a user comes up with a new device for the environment, she may not be able to set it up to work immediately with the environment and share it with the people she trusts. These tasks have to be done by a group of system administrators who configure the devices and create accounts in the centralized service.

Furthermore, the user depends on the server to be authenticated. A principal can only be authenticated to use a resource not belonging to him if it is online with the server. What would then happen when a user needs to work with a device belonging to another user and both are disconnected from the centralized service?

The main contribution of this paper is an architecture that avoids explicit authentication and eases sharing, and the implementation of such system.

The rest of this paper is organized as follows. In the next section, we further introduce our approach to the problem. An illustrated problematic scenario and a possible solution is presented in section 3. In section 4 we present the architecture of the system. Restrictions and countermeasures against attacks will be presented in section 5. Section 6 provides a short description of the SHAD prototype's implementation and its integration with the smart space. Some aspects of the described architecture are discussed in section 7. In section 8 we discuss related work. Finally, section 9 presents the conclusions and future work.

2 Introducing the SHAD approach

In this paper, we present SHAD , a Peer-to-Peer Single Sign-On architecture for the ubiquitous space. It follows the ideas stated in [36] in order to offer two services:

1. SHAD offers **Single Sign-On for the ubiquitous environment**. The user has to authenticate himself only once in order to access any service from any of his machines. This way, users can keep the illusion that *the ubiquitous environment is a unique system, not a set of interconnected systems.* A SHAD main agent serves the secrets (keys, passwords, etc.) to all other machines belonging to user. Any machine owned by the user can run the SHAD main agent. The

main agent is designed for running in a mobile device that can be always carried by the used. SHAD is part of the UbiTerm, which also offers other services (such as controlling user's activities). In SHAD, users can set different policies in order to distribute their secrets. The policies are based on attributes assigned to the secrets. Therefore, the security level of the secrets is up to each user. This aspect will be thoroughly described in section 5.

2. SHAD permits users to **share their belongings in a non-obtrusive and easy way**. Every machine has an owner, and the owner has a UbiTerm to control leasing. When two humans meet and agree on having mutual trust, they have to *pair* their SHADs to provide the base for further trust relationships. This way, users can administrate their own devices and share them in a natural way, doing it the same way used for real life relationships.

Once the SHADs are *paired*, any device owned by one of the humans can be used by the other (of course, depending on the policies and attributes set up by the owner). Note that users only have to pair the SHADs to potentially lend some or all their belongings to the other side according to a policy.

The idea behind our scheme is similar to the use of PIN numbers to pair Bluetooth devices: we pair users pairing their SHADs. The pairing process must be done only once at configuration time and it can be automatized. In this pairing process, the SHADs negotiate a shared secret that will be stored in the user's secret store. This shared secret permits SHADs to authenticate each other, negotiate session keys and share devices. Note that users only have to pair their SHADs in order to share any of their devices. The pairing relationship between two SHADs (that is, between two users) is non-transitory.

In some cases, the owner of a device may not be a real user. For example, the owner of terminals located at a laboratory may be a virtual user named 'lab'. All the persons allowed to use these terminals have to be paired (i.e., share a secret) with 'lab'. The agent for user 'lab' could be running on a server located anywhere in the environment.

There may be problems because of using a mobile device in order to authenticate users, but this will be discussed later in section 7.

3 Elaborating the Problem

3.1 A Problematic Scenario

User Bob owns three devices: a Pocket PC, a laptop and a workstation. When he arrives to his office, he has to boot both the laptop and the workstation. At boot time, he has to type the login name and the password in both devices. He also has to enter the password in his Pocket PC.

Suppose that Bob has some of his documents stored in a remote file server. In order to access it he has to mount the remote file system on the three devices. To mount a remote file system (for example using NFS or SMB), he is forced to authenticate himself again. That implies supplying three more passwords.

Later, Bob has to attend a lecture in a near room that is equipped with general purpose terminals. During the lecture, Bob needs to check his e-mail. He

would like to use one of terminals available in the room for that purpose. Thus, he has to be authenticated in this terminal and he has to supply yet another password. Moreover, he has to supply the POP3 password to be authenticated on the mail server.

A situation like this one happens often at any office environment or university department.

This makes the system quite hostile for humans. Note that since Bob arrived to the office, he has been forced to explicitly authenticate himself for eight times. If Bob had used existing OSS software, he would have had to authenticate himself at least for four times (one authentication per machine). Therefore, for the user that is not a Single Sign-On system!

3.2 Using SHAD to fix the problem

When Bob arrives to the office, he enters a passphrase in his Pocket PC (his UbiTerm). In this very moment, a SHAD agent starts in the Pocket PC and reads a file from an SD Card inserted in the device. Using the passphrase typed before, the agent decrypts the file (encrypted with a strong algorithm). Then, it reads all the secrets belonging to Bob (passwords, keys, etc.) and stores them in the agent's process memory (which is not accessible from other processes or machines).

When Bob boots the laptop, a SHAD agent is executed on it. Next, the laptop operating system asks its local agent for authentication. Then, the laptop's agent discovers that another one belonging to the same user is running (the Pocket PC agent) and it realizes that this agent has all the secrets of Bob in it. Instead of prompting for the password , the laptop's agent asks the Pocket PC's agent for the secret needed. This time, Bob's Pocket PC beeps and asks for confirmation. Next, Bobs presses a button in the Pocket PC in order to confirm the laptop booting [1]. The boot process of the workstation is analogous. When the Pocket PC needs to mount the remote file system, the local agent provides the password.

Later, Bob attends the lecture and he wants to check his mailbox in a terminal. He is carrying his Pocket PC, which is executing the SHAD agent. This terminal belongs to Alice, the person in charge of the lecture room. Assume that some time ago, Bob and Alice agreed on mutual trust (that is, they trust each other) and they interchanged a secret. This secret was stored in their SD Cards.

The terminal booting for Bob realizes that a person claiming to be Bob is the one closer to the terminal because he is wearing an RFID tag with his user id [2]. Of course, at this time Bob has not been authenticated yet. The terminal's agent makes contact with Alice's SHAD agent (for example running on Alice's laptop) and with Bob's SHAD agent. Bob's SHAD agent notifies Bob that somebody is

[1] As we will describe later in section 5, the system is highly customizable and there are different kinds of confirmation methods and several countermeasures in order to avoid not authorized password forwarding. In this example, Bob has to confirm the sending of this password by pushing a button.

[2] Or by any other means.

trying to boot a terminal in his behalf, and Bob confirms the operation just by clicking a button. Alice's SHAD agent on the other hand is configured to grant the lending without explicit confirmation [3]. Thereafter, Alice's SHAD agent and Bob's SHAD agent start a protocol using their shared secret to authenticate each side.

If the protocol is successful and Alice's SHAD agent allows the leasing, the terminal will be notified to allow Bob to use it. When Bob tries to download his mailbox, Bob's SHAD agent will provide the POP3 password to the terminal's agent (which is now running in behalf of Bob). Note that Bob has not been required to type any password (nor login, thanks to his badge) in the terminal, he has only been required to confirm the operation by clicking a button. In addition, if the the agents had taken advantage of any context infrastructure of the ubiquitous environment, the explicit confirmation could have been avoided.

4 Architecture

The architecture for SHAD is shown in Fig. 1. Basically, every device has an owner, usually a human. All devices owned by a user run a SHAD agent, that is a Peer-to-Peer program that can adopt the main agent role or the plain agent role.

The **main agent** role is adopted by the agent that provides secrets to other agents. This agent has some or all of the secrets of the user. There can be only one main agent for a user. When this agent starts, it obtains the secrets from a secured repository. In the scenarios described in section 1, the agent reads the secrets from a SD Card inserted in the device. Of course, it is not the only way to obtain the secrets store. It can be obtained from a local storage device or from a remote centralized service. Other secrets could be supplied by the user to the main agent when needed and they can be provided to other agents (in accordance to the policies and attributes associated to them). In addition, the main agent provides authorization for using the devices belonging to the user. The main agent is intended for running in a mobile device (UbiTerm).

The **plain agent** role is assumed by the rest of security agents. The plain agents act like clients and ask the main agent for the secrets needed in its local device. If any local program requests a secret used before in this session, the plain agent does not ask the main agent because it already has the secret. In contrast, if the plain agent does not have the secret (it has not been required before in this machine), it asks the main agent. The main agent serves the secrets to the plain agent in accordance to the policy set up by the human.

4.1 External Entities

Figure 1 depicts the architecture presented before. We can identify the following external entities: the user, application programs, and the secret store.

[3] Note that Alice can choose between different options to secure this lending. They are explained in section 5.

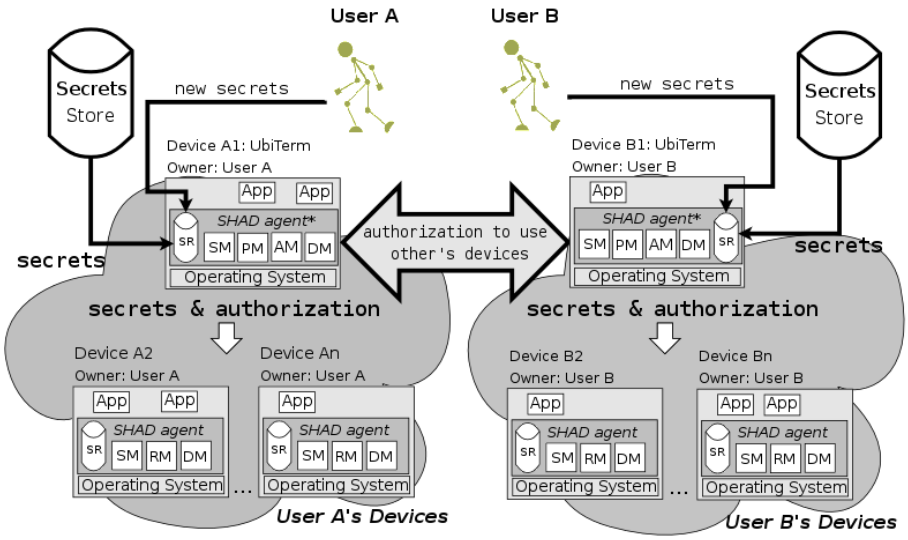

Fig. 1. SHAD's global architecture. SHAD agents marked with a * are main SHAD agents.

The **user** gives new secrets to the agent. These secrets are usually given to the main agent. The user can supply secrets to any plain agent as well. In this case, the secret is only available in the machine executing this plain agent.

But **application programs** are the clients for the agent. When an application needs a secret, it asks the local agent for it. The local agent makes contact with the main agent in order to get the secrets required by local applications. If it cannot obtain these secrets, it asks the user to supply them using any input device. In this particular case, the agent behaves just like a traditional SSO agent does. Application programs operate with SHAD through a virtual file server interface, following the same approach used in most Plan 9 operating system services [30].

The **secrets secure store**, where the user secrets may be kept. This repository is strongly encrypted. It can be local or remote, depending on the user's necessities. It's functionality mimics Plan 9's secstore [6].

4.2 Internal Entities

There are some elements in the architecture implemented by the agents. The **Secrets Module (SM)** serves the secrets to the user's applications. It also reads new secrets from input devices like keyboards. This module is active in all agents. In fact, this module is a virtual file server running in user's space. The **Provider Module (PM)** is the one providing keys through the network. It also provides authorization when other users want to use a device belonging to its owner. This module is active exclusively in the main agent. The **Announce**

Module (AM) responds to look-up messages asking for the main agent. To do that, it is listening at a well known port for broadcast messages. This process is only active in the main agent. The **Retrieve Module (RM)** is the process that sends requests to the main agent to receive a secret needed by any local application. It is only active in plain agents. The **Discover module (DM)** is the module in charge of discovering fellow main agents. It is active in all agents. Finally, the **Secrets ring (SR)** is the data structure in which the user's secrets are stored. It resides in main memory and has to be protected. This memory should not be accessible by any process but the ones cited above.

4.3 SHAD Keys

SHAD keys must be stored in the secrets secure store of the owner, together with the secrets for common services.

As stated before, two users that trust each other must have a secret shared key. We refer to these keys as **SHAD pairing keys**. Main agents can authenticate each other by using a SHAD pairing key.

In order to support our architecture, a secret key must be kept stored in each device. This secret key is created and stored together with the identification name of the owner of the device at configuration time. Usually, this is done when the device is first installed. This kind of secret is named **SHAD terminal's secret**. It permits the main agent to authenticate the machines owned by the user.

When a user needs to add a new device to the environment, she does not have to make contact with any administrator. She only must to follow two steps: First, she must create a key for the new device and store it in her secrets store. Second, she must assign the key to the device, for example by writing it in the NVRAM of the device.

We assume that an attacker does not have physical access to the hardware. Physical access to the hardware implies a great risk, because the secret stored in NVRAM could be compromised and the attacker could impersonate this machine and access to the owner's secrets.

Note that when an attacker has physical access to the hardware, securing it is pointless. This principle is also known as *The Big Stick Principle* [38, Chapter 4]. In this case, the adversary can manipulate it, steal the disks containing all the information, modify the software and the operating system to be malicious or literally burn and destroy the machine. Other systems are based in the same assumption, see for example the GAIA[32] security architecture[33]. Plan 9 [30] authentication servers reads their private key from NVRAM at boot time too.

4.4 Protocols

In order to provide the secrets service and the sharing service, SHAD uses three different protocols [4]. The **Main agent discovery and machine leasing protocol** is used to find the main agent belonging to the user. In addition, this

[4] Due to space limitations, these protocols are not described in this paper

protocol allows a human to use a machine not belonging to him. The **Secrets delivery protocol** is used to provide the secrets to plain agents. Finally, the **Other's main agent discovery protocol** is used to discover the main agent of other users.

5 Restrictions and Countermeasures

The secrets stored in the secured repository have some attributes that are used to set access control policies and restrictions. We describe them here.

SHAD terminal's secrets have two special attributes that restrict the use of terminals. An attribute named **forleasing** specifies if this terminal can be lent to other user or not. The **needconfirm** attribute forces the booting of a machine to be explicitly confirmed by the owner. If this attribute has a positive value or it is omitted, the booting always has to be confirmed by the owner.

The *user's common secrets* (passwords, keys, etc.) have another attributes. The **noremoteaccess** attribute means that the secret cannot be sent outside the local agent in any case. The **needconfirm** attribute forces explicit confirmation in order to share the secret. The user can confirm operations just by pressing a button on the machine running the main agent.

The **userlocation=***location* means that the secret can only be sent if the user is at the specified location. Note that if the **needconfirm** is set, the operation also needs to be confirmed by the user. This location information is provided by a external context infrastructure[5]. If the context infrastructure is not available, the secret cannot be sent. The attribute named **clientlocation=***location* works in a similar way, but taking into account of the location of the machine running the plain SHAD agent, that is requesting the secret. The **samelocation** attribute requires the user and the client machine to be at the same physical location. The attribute **accesiblefrom=***machine* makes the secret available only from a plain agent running on the specified machine.

All these attributes can be combined in order to restrict the access to the secrets. The user must evaluate the trade-off between comfort level and security level from his own perspective and set the attributes according to it.

The more restrictive is the access to a secret, the more obstructive the system is, because required secrets have to be supplied by the user. The same occurs with confirmations.

When the main agent sends a secret (with or without user confirmation), it always prints a warning about it and all these messages are saved in a log file.

The user can set a timeout to block the main agent if the machine where it runs is idle for a given period of time. When the timeout expires, the secrets stored in main memory are deleted and the agent shuts down. In order to active the main agent, the user must introduce the passphrase again. Then, the secrets

[5] Securing the context infrastructure is out of the scope of this paper. We assume that context data is correct and reliable, but not fault tolerant. If a user does not trust this infrastructure, it suffices to avoid the *location attributes. The trade-off, as we said, is made by the user.

are retrieved from the secret store and decrypted. If the user loses the UbiTerm, there is a customizable window of vulnerability.

In addition, the UbiTerm's terminal key is kept encrypted, because it can be decrypted using the (unique) explicit confirmation provided by the user. This way, the UbiTerm's terminal key is not compromised when the UbiTerm is lost.

6 Implementation and Integration

A SHAD prototype has been implemented for the Plan B Operating System [16, 17]. SHAD is intended for an ubiquitous environment where all services are exported as file systems. Therefore, all these systems are accessed by the file service protocol and SHAD provides authentication for this protocol. As a consequence, SHAD provides SSO for accessing all the elements in this ubiquitous environment.

We have chosen Pocket-PC devices with Bluetooth, WI-FI and GPRS capabilities to host the UbiTerm. As other authors [12], we think that authentication devices must offer other services to users in order not to be ignored or forgotten. In our system, the UbiTerm is the main security server, a user's activities monitor and a general purpose handheld device.

SHAD is a Factotum[21] derivative. Factotum is the Plan 9 [30] security agent. It provides SSO in one machine through a file server interface. Basically, our implementation of SHAD uses Factotum as the secrets module (SM) described in previous section and illustrated in Fig. 1. Therefore, applications using SHAD do not depend on any kind of middleware or frameworks. To add secrets to a Factotum, the user only needs to write tuples in a file. The tuples contain attributes for the secret, for example the protocol or the server. SHAD attributes are included in these tuples. For the secrets specific for SHAD (terminal's secrets and pairing keys), we have added a new protocol to Factotum, which is named `SHADP`.

The prototype uses the AES symmetric algorithm in CBC mode in order to encrypt the communication between agents. Therefore, SHAD pairing keys and terminal's secrets are AES keys. In addition, it uses SHA-1 in order to obtain hashes and create digital signatures together with AES.

As we stated before, the SHAD terminal's secret is stored in the NVRAM of the machine. We are looking for hardware alternatives to the NVRAM. A solution would be to use any tamper resistant hardware using challenge-response methods (e.g., Smartcards [31]) in order to authenticate machines.

Last but not least,to avoid the need for obtrusive identification wherever feasible (and to implement some restrictions) SHAD takes advantage of a real context and location data infrastructure [15, 18].

7 Discussion

The problem of using a personal server to authenticate users is that the security data depends on a physical object, and it might be stolen or lost. Nevertheless,

in real life humans already depend on physical objects that may be stolen or lost. For example, credit cards and official documents like the driving license or the social security card. When humans lose these very important physical objects in real life, they immediately call the bank office or the police. Likewise, if a user loses the device with his secrets, he should revoke these secrets.

In general, security is a trade-off between safety and costs[35]. In this case, users have to accept the trade-off between comfort and the risk of losing the physical object that contains their secrets. In other words, users must face the risk of losing the device that provides the non-obtrusive authentication

In addition, there are several ways to warn users about stolen keys For example, compromised secrets can be announced in an centralized service which has to be periodically visited by system users.

SHAD implements some countermeasures against the possible security risk derived of losing this device, such as confirmations and timeouts. The system is highly configurable and flexible. Users preferring comfort to higher security can customize SHAD in order to serve all passwords and rarely ask for confirmation. On the other hand, users preferring extra security instead more comfort can disable the leasing for most important passwords and require confirmation for all requests. In real life, users prefer a combination of comfort and security and should configure SHAD according to their perspective. We have to keep in mind that most systems are vulnerable if the user makes wrong choices. For instance, password based systems are vulnerable if lazy users set a blank password in order not to type at log in.

8 Related Work

Several *password tools* and *SSO systems* have been proposed in research and commercial products. As far we know, none of them provided single sign-on to users working with multiple devices simultaneously.

There are *password tools* that permit the user to copy his passwords and paste them in the applications. The passwords are strongly encrypted and stored in the local file system. When the user needs to authenticate, she must select the password from a GUI, copy it and paste it in the application. See for example Password Safe[34] or its ports to other systems (MyPasswordSafe[2] or Pwsafe[3]).

This kind of systems makes authentication more comfortable than password typing, but they are still obtrusive. SHAD is not, because it can provide secrets without human interaction.

Simple *SSO agents* permit single sign-on for an single service, for example the SSH agent Web browsers, such as Netscape, remember passwords and provide SSO for web applications. Other SSO systems provide single sign-on for different services, for example SecureLogin Single Sign-On[7], Focal Point[8], and Factotum[21]. These systems make authentication non-obtrusive for traditional schemes, where the human uses a workstation and accesses all distributed services through the network. These SSO systems provide passwords to applications

to access services without user interaction. They usually store the secrets on a central server. The agents running in the user's workstation send requests to the centralized service providing the secrets. In these systems, the agents normally have a password cache in order to support disconnections.

But *SSO systems* have two important problems. First, the clients **depend on centralized services** (the password provider or the secret store server). We consider that centralization is not appropriate for ubiquitous environments because there are cases when users need to be authenticated and they are disconnected from the centralized server. Second and most important, if the human needs to use several devices simultaneously, **there is no single sign-on**. As a consequence, these systems do not full fill the requisites of scenarios like the one described in section 3. We centralize the service, but instead of a global server we have personal authentication servers. The personal server provides all authentication related with its owner. Therefore, SHAD works well in locations where several principals can communicate but there is no connection to the rest of the system. In addition, SHAD provides SSO even when user is working with multiple machines at same time.

Factotum[21] together with Secstore[6] provides SSO from a single workstation following a similar approach. In addition, a Factotum agent can be used from other machines by importing its file server. But it does not provide single sign-on for multiple devices, because the machine importing the remote Factotum file system had to be authenticated before. Instead, SHAD agents cooperate to provide SSO in all terminals running on behalf of the user. Also, SHAD adds a new protocol that permits humans to set trust relationships and share their terminals easily. Factotum is independent of frameworks and complex middleware because of its file server approach. SHAD is a Factotum derivative and adopts the same scheme.

CorSSO [26] removes all authentication mechanisms from the application servers and relocates them at multiple authentication servers. The application servers store threshold cryptography [22] keys in different authentication servers in order to make the client not to depend on a single centralized server. When a client needs to be authenticated by the application server, it has to retrieve a subset of these keys from the authentication servers. To make contact with the authentication servers, the client authenticates himself always with the same identity (i.e. the same password) hence the SSO in CorSSO. It solves the problem of centralization by making users not to depend on a single authentication server. However, it does not solve the problem of authenticating users at isolated locations where they are disconnected from the rest of the system. In addition, CorSSO's scheme requires changes in both client and server applications. SHAD, as Factotum, places the authentication mechanism out of the applications. Finally, CorSSO does not allow SSO when the user works with multiple machines simultaneously. As in other SSO systems, the user has to authenticate himself at least once per machine.

Machine sharing is normally faced using centralized servers like Kerberos[40] and Sesame[27] that provide authentication to allow principals to use terminals.

These systems depend on centralized services and are hard to administer. They block ad-hoc interactions with guest users. In SHAD's approach, when a new user comes to the environment, she only has to pair her UbiTerm with people she trusts.

The use of *authentication devices* has been proposed in several previous works, but they are normally used to authenticate users for specific purposes. For example, Beaufour et al. [19] use a bluetooth enabled mobile phone to unlock office doors in an ubiquitous environment. Corner et al. [20] use a mobile device to protect the applications memory while the user is away. Shared PC[25] uses a Smartcard containing a private key to authenticate the user and restore the session from public shared terminals. Other systems use a CryptoToken[28], that is a USB device that has stored the secrets of the user and provides SSO in all machines. When the user needs to work with a machine, she connects the CryptoToken to it. This approach is only a bit less obtrusive that typing passwords, but it is still a burden for the user. In addition, not every device has USB ports. Finally, it does not support concurrent authentications in different machines, because the user only has one CryptoToken.

In contrast, we propose the use of an authentication device named UbiTerm that enables a general purpose security architecture, not to solve a specific problem. None of these works provides SSO for common services as SHAD's UbiTerm does. Also, SHAD permits authentication between users in order to allow them to share resources.

Resurrection Duckling[37] proposes *secure transitory associations* between devices and humans in an ubiquitous environment. We propose instead using non-transitory associations between devices representing humans. We also propose transitory associations between humans and devices not belonging to them. Our architecture is an attempt to extend this approach and create a general purpose security architecture.

Other architectures use *trust estimations* based on reputation, recommendation and experience. See for example AAPR [29] and the works of Abdul-Rahman et al. [9] and Aberer et al. [10]. The SHAD approach does not try to simulate human behaviors, it offers an architecture that allows humans apply their real trust relationships instead in order to share their belongings.

Other security schemes have been proposed for *ubiquitous environments*. Most of them depend on complex middleware architectures and are highly centralized. In general, middleware based architectures make developers depend on specific platforms or languages.

Some works [11, 13, 14] related with GAIA Smart Spaces [32] were heavily based in centralized security schemes like Sesame and Kerberos, therefore they have the same problems cited above. In short, they have been designed following a classical client/server scheme in order to create restrictive and isolated smart spaces. SHAD proposes a security scheme for open environments where users cooperate and share resources with others. In Cerberus[14], users can authenticate using different methods (passwords, smart badges, etc.) depending on them confidence level and security level estimated from context information.

This security level is estimated by an inference engine together with a knowledge base. In addition, Muthadi et al. propose in their work[12] a watch with both processing and communication capabilities in order to authenticate users in this architecture.

Cerberus is very flexible and it provides a fine grained access control, but it depends on the inference engine and the authentication server. SHAD does not depend on central services and it is not designed for restrictive client/server schemes. We propose an architecture that can be used for both Peer-to-Peer and classic client/server settings. SHAD can handle client/server settings through creating non-real users and using servers as their principal agent. In addition, SHAD does not force developers to use a specific platform or middleware thanks to its file server interface.

Similarly, HESTIA[23] proposes a middleware architecture that adds security, fault tolerance and privacy to services. HESTIA proposes a intermediate layer between clients and servers in order to provide these security and privacy services. This architecture is client/server oriented too. Therefore it is not suitable to be used in a Peer-to-Peer setting.

9 Conclusion and Future Work

The main contributions of this paper are: (i) the description of the SHAD architecture; (ii) a way to provide **real Single Sign-On** in ubiquitous environments where users work with several machines at same time; (iii) a non obtrusive way to secure device sharing in a smart space; and (iv) a prototype implementation of the SHAD architecture.

SHAD does not require complex administration, and it is independent of centralized services. Last but not least, it does not depend on any kind of middleware or framework, because it relies on distributed file system technology. All these properties make SHAD different from other architectures presented in the literature.

We are adding new Peer-to-Peer functionalities to plain SHAD agents. For example, a plain agent could serve some secrets to other plain agent when the main agent is not available. We are currently working on these new mechanisms.

Future work will be centered in revocation mechanisms. This is a non-trivial problem that has to be taken into account. In addition, some fault tolerance functionalities could be added to the discovery process in order to support crashes of the main agent.

References

1. Ibutton. http://www.ibutton.com.
2. Mypasswordsafe manual. http://www.semanticgap.com/myps/.
3. Pwsafe password database. http://nsd.dyndns.org/pwsafe/.
4. Rsa security. http://www.rsasecurity.com.

5. *The Network Information Service*. Sun Microsystems, 1990. in System & Network Administration, SunOS 4.1 manual set.
6. *Plan 9 Programmer's Manual, Section 1: Commands*. AT & T Bell Laboratories, Murray Hill, NJ, fourth edition, 2002. http://www.cs.bell-labs.com/magic/man2html/1/secstore.
7. *SecureLogin Single Sign-On White Paper*. Protocom Development Systems, 2003. http://www.protocom.com/html/whitepapers/.
8. *Focal Point Evaluator's Guide*, 2004. http://www.okiok.com/index.jsp?page=Focal+Point.
9. A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *Proceedings of Hawaii International Conference on System Sciences 33*, 2000.
10. K. Aberer and Z. Despotovic. Managing trust in a peer-to-peer information system. In *Proceedings of the 9th International Conference on Information and Knowledge Management (CIKM)*, 2001.
11. J. Al-Muhtadi, M. Anand, N. D. Mickunas, and R. Campbell. Secure smart homes using jini and sesame. In *Proceedings of the 16th Annual Computer Security Applications Conference*, New Orleans, USA, 2000. IEEE Computer Society.
12. J. Al-Muhtadi, D. Mickunas, and R. Campbell. Wearable security services. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, 2001.
13. J. Al-Muhtadi, A. Ranganathan, R. Campbell, and N. D. Mickunas. A flexible, privacy-preserving authentication framework for ubiquitous ccomputing environments. In *Proceedings of IWSAEC 2002*, 2002.
14. J. Al-Muhtadi, A. Ranganathan, R. Campbell, and N. D. Mickunas. Cerberus: A context-aware security scheme for smart spaces. *IEEE International Conference on Pervasive Computing and Communications*, 2003.
15. F. J. Ballesteros, G. Guardiola, E. Soriano, and K. Leal. Traditional systems can work well for pervasive applications. a case study: Plan 9 from bell labs becomes ubiquitous. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications*, 2005.
16. F. J. Ballesteros, E. Soriano, K. Leal, and G. Guardiola. Plan b: An operating system for ubiquitous computing environments. In *Proceedings of the 4th IEEE International Conference on Pervasive Computing and Communications*. 2006.
17. F. J. Ballesteros, E. Soriano, G. Guardiola, and K. Leal. Plan b: A pervasive computing environment without middleware designed for programmers. *To appear*. 2006.
18. F. J. Ballesteros and E. Soriano. The ls smart space for programmers. *Submitted for publication. Also at http://lsub.org.*, 2005.
19. A. Beaufour and P. Bonnet. Personal servers as digital keys. *Second IEEE International Conference on Pervasive Computing and Communications*, 2004.
20. M. Corner and B. Noble. Protecting applications with transient authentication. *In First ACM/USENIX International Conference on Mobile Systems, Applications and Services*, 2003.
21. R. Cox, E. Grosse, R. Pike, D. Presotto, and S. Quinlan. Security in plan 9. In *In Proceedings of the 11th USENIX Security Symposium*, 2002.
22. Y. Desmedt. Threshold cryptosystems. In *Proceedings of Advances in Cryptology - Auscrypt'92, LNCS 768*, 1993.
23. R. Hill, J. Al-Muhtadi, R. Campbell, A. Kapadia, P. Naldurg, and A. Ranganathan. A middleware architecture for securing ubiquitous computing cyber infraestructures. *IEEE Distributed Systems Online*, 2004.
24. L. Hong and A. K. Jain. Integrating faces and fingerprints for personal identification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1998.

25. S. Izuka, K. Uwazumi, K. Nakahama, S. Nakajima, and K. Ogawa. Secure pc environment roaming technology for ubiquitous office. *Workshop on Security in Ubiquitous Computing, Ubicomp 2003*, 2003.
26. W. Josephson, E. G. Sirer, and F. B. Schneider. Peer-to-peer authentication with a distributed single sign-on service. In *Proceedings of the International Workshop on Peer-to-Peer Systems*,2004.
27. P. Kaijser, T. Parker, and D. Pinkas. Sesame: The solution to security for open distributed systems. *Computer Communications, vol. 17, pp. 501-518*, 1994.
28. H. Kopp, U. Lucke, and D. Tavangarian. Security architecture for service-based mobile environments. In *Proceedings of the 2nd International Workshop on Middleware Support for Pervasive Computing, IEEE PerCom 2005*, 2005.
29. H. Lee and K. Kim. An adaptive authentication protocol based on reputation for peer-to-peer system. *Symposium on Cryptography and Information Security (SCIS) 2003*, 2003.
30. R. Pike, D. Presotto, K. Thompson, and H. Trickey. Plan 9 from bell labs. In *In Proceedings of the Summer 1990 UKUUG Conference*, 1990.
31. W. Rankl and W. Effing. *Smart Card Handbook*. John Wiley and Sons, second edition, 2000.
32. M. Roman, C. K. Hess, R. Cerqueira, A. Ranganat, R. H. Campbell, and K. Nahrstedt. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing, vol. 1, pp. 74-82, 2002*, 2002.
33. G. Sampemane, P. Naldurg, and R. Campbell. Access control for active spaces. *In Annual Computer Security Applications Conference (ACSAC2002)*, 2002.
34. B. Schneier. Password safe. http://www.counterpane.com/passsafe.html.
35. B. Schneier. *Beyond Fear: thinking sensibly about security in an uncertain world*. Copernicus Books, New York, NY, 2003.
36. E. Soriano. Shad: A Human Centered Security Architecture for Partitionable, Dynamic and Heterogeneous Distributed Systems. In *Proceedings of the 5th ACM/IFIP/USENIX International Middleware Workshops*. 2004.
37. F. Stajano. The resurrection of duckling - What next? In *Proceedings of the 8th International Workshop, LNCS 2133*, Cambridge, UK, 2000. Springer-Verlag.
38. F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, 2002.
39. F. Stajano and J. Crowcroft. The butt of the iceberg: Hidden security problems of ubiquitous systems. In *Ambient Intelligence: Impact on Embedded System Design*. Basten et al., 2003.
40. J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of the Winter 1988 USENIX Conference*,1988.
41. M. Weiser. "The Computer for the 21st Century". *Scientific American*, 265(3):94–104, September 1991.