

Secure Smart Homes using Jini and UIUC SESAME

Jalal Al-Muhtadi
*Department of
Computer Science,
Univ. of Illinois at
Urbana-Champaign,
almuhtad@uiuc.edu*

Manish Anand
*Department of
Computer Science,
Univ. of Illinois at
Urbana-Champaign,
manand@uiuc.edu*

M. Dennis Mickunas
*Department of
Computer Science,
Univ. of Illinois at
Urbana-Champaign,
mickunas@uiuc.edu*

Roy Campbell
*Department of
Computer Science,
Univ. of Illinois at
Urbana-Champaign,
rhc@uiuc.edu*

Abstract

In this paper, we discuss our approach to constructing a dynamic and secure “Smart Home” environment and tackling the challenges associated with it. We envision a Smart Home as an active environment populated with smart, dynamically configurable consumer devices capable of interacting with humans and other smart devices. In such a dynamic and active environment, there is a great need for an agile, lightweight, distributed security mechanism. This security mechanism needs to be programmable and able to evolve as rapidly as the environment itself. Yet, this mechanism should be able to adapt to environments with scarce resources. We present Tiny SESAME to meet these challenges while utilizing Jini™ technology from Sun Microsystems to handle the common parts of distributed devices. Tiny SESAME is a lightweight, component-based, Java-implementation of a subset of SESAME. SESAME is an extension to Kerberos that supports public key technologies, access control, and delegation of access rights. We discuss our Tiny SESAME and how it could be integrated with handheld and consumer devices.

1. Introduction

With the current advancements in computing power and network technology, the concept of fully automated “smart” homes is not far fetched. Jini™ technology from Sun Microsystems provides the necessary facilities for creating “network plug and play” devices, which when connected to a network, announce their presence and enable network users to remotely connect to them [Edw99]. Once all devices in a household are automated and connected through a network, it becomes important to consider issues of security, authentication and access control. For example, parents may want to ensure that children will not be able to access the VCR after 7 PM on

schooldays. Likewise, children under 17 should not be able to view an R-rated movie on the DVD-player. While at work, a person would like to be able to check and set the status of his or her home security alarm remotely, or lock/unlock the front door; but certainly, strangers should not be able to access these services, or even to eavesdrop on the communication channel.

Since Jini™ provides a set of APIs, protocols, and conventions that can handle the common parts of distributed devices, building a Smart Home environment is fairly straightforward. In this paper, we are concerned about constructing a dynamic and secure Smart Home environment and tackling the challenges associated with it. Most importantly, our goal is to introduce a security mechanism that can be easily embedded into lightweight distributed devices with significantly different capabilities. We argue below that existing mechanisms, particularly Java security, Kerberos, and SESAME, are either inadequate or too heavyweight for this task.

We have simulated a dynamic Smart Home environment, for which we built a few smart consumer devices using Jini, and connected them via a network. To secure the environment we developed a security mechanism called *Tiny SESAME*, a lightweight subset of *SESAME* [Ses] and [Kai94]. *Tiny SESAME* allows Jini services to authenticate users, control access, and encrypt sensitive data. Further, a GUI interface is provided that graphically displays all the networked devices and allows a user to log-on (if required by the device) and to configure the devices. The GUI interface is capable of detecting new smart devices dynamically once they are plugged in, and is able to download the properties and capabilities of such devices on the fly. Likewise, the GUI interface can detect when a device is unplugged from the network, and subsequently can automatically remove the device from the list of those available.

The remainder of this paper is organized as follows. Section 2 provides background that motivates the need for a lightweight security mechanism.

Section 3 describes the Tiny SESAME's architecture. In Section 4 we present our notion of smart devices, and we describe how mobile code and our Smart Home client are used to control such devices. In Section 5 we discuss the integration of Tiny SESAME with the Smart Home environment. We conclude in Section 6 with some directions for future work.

2. Background

2.1 The need for security

Having Smart Home devices accessible and configurable over the network is very useful. However, there must be some security enforced to keep unauthorized personnel away. Further, there is a need to authenticate a user before configuring a device. It is also desired in some scenarios that some level of access control be enforced especially for critical devices such as the home security alarm, or the front gate. The challenge goes beyond this. Since the Smart Home environment is dynamic in nature, the mechanism to secure the environment has to be agile and distributed. Additionally, the mechanism needs to be programmable and adaptable to different consumer devices.

The current Jini's security model is based on JDK 1.2 security. This security model provides good security against ill-behaved downloaded code, through sandbox settings, byte verification and type safety. Consequently, this model can be employed to provide straightforward and plain security in a distributed environment. However, this security model lacks support for distributed security services, authentication and authorization, as Java authentication facilities were immature at the time of this writing. Further, the existing security model fails to scale well in large distributed systems. Our objective is to base network authentication and security on a commercially available security infrastructure, thereby granting us flexibility, scalability, and the ability to achieve secure interoperation between different devices regardless of their platforms and programming environments.

2.2 Security mechanisms

To meet the challenges created by securing distributed systems and authenticating the entities thereof, Kerberos [Neu94] was developed at MIT [Ker] during the mid 1980's to provide security services for the Athena campus-computing network. However, Kerberos has certain limitations. First, it is based on symmetric cryptography, so it does not scale well for large open distributed systems [Muf94]. Additionally,

it lacks support for access control [Mcm95].

SESAME (*A Secure European System for Applications in a Multi-vendor Environment*) [Ses], [Mcm95] and [Kai94] is an extension to Kerberos that provides additional services. These services include using digital signatures for login, preventing offline dictionary attacks, handling of access control privileges for users and supporting different key management and distribution schemes. Moreover, to allow application developers to utilize its security services, SESAME adopts the Internet standard GSS-API (*Generic Security Services Application Programming Interface*) (see RFC 1508 [Lin93], and RFC 2078 [Lin97]). GSS-API is a standard programming interface for generic security services. SESAME was standardized by ECMA [Ecm96a] and [Ecm96b]. Although a full description of the SESAME architecture is beyond the scope of this paper, in Section 3 we describe a lightweight subset of the SESAME for use in handheld and consumer devices. For a complete description of SESAME, the reader is referred to [Kai94], [Mcm95], [Kai98], [Ses], and [Ash99].

3. Tiny SESAME

SESAME with its various components requires relatively high resources in terms of memory and processing power, which exceed the typical capacity of mobile or consumer devices. *UIUC SESAME* [Cha99] is a portable version of SESAME implemented completely in Java. *Tiny SESAME* is a subset of UIUC SESAME that supports authentication, simple encryption, integrity checks and RBAC (Role-Based Access Control). To keep it "lightweight", Tiny SESAME employs a component-based architecture. It does not support restricted delegation, PV/CV, auditing. Current version of Tiny SESAME is capable of authenticating entities through password-based authentication as of yet. A GSS-API is implemented in Java [Muh99], which allows Jini services to interface with the security services. Figure 1 shows an overview of Tiny SESAME.

3.1 Client side

The client side is where a Smart Home user interacts with the Smart Home client (described in Section 4) to access and configure the smart devices. The Smart Home client resides on some *user sponsor*. The user sponsor could be a personal computer, a handheld, or any device capable of running a web browser. The SACM (Secure Association Context Manager) provides data integrity and confidentiality services. The GSS-API atop of SACM provides Java programs

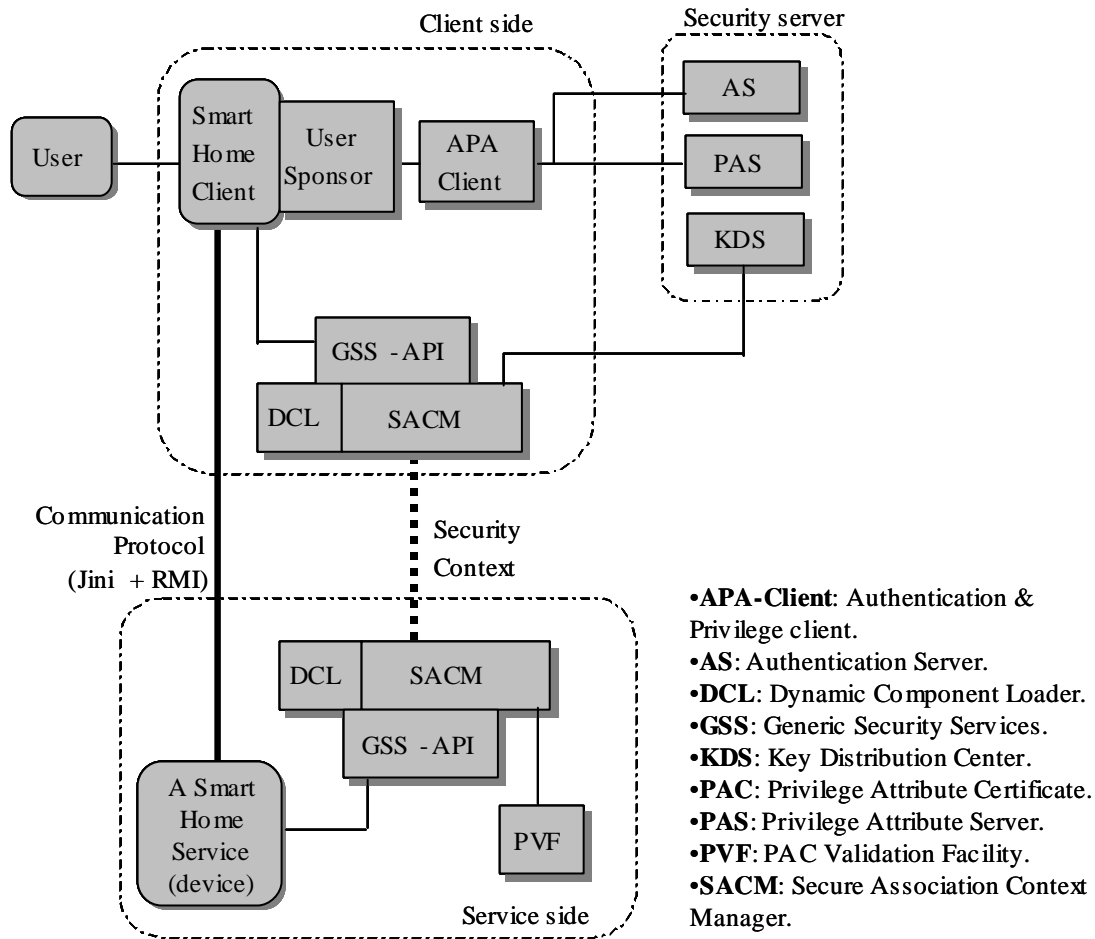


Figure 1: Tiny SESAME

with a standard interface to access SACM services. Since SESAME's protocols are rather complex and resource demanding, in Tiny SESAME different protocols and encryption routines are implemented as separate software components. These components are loaded dynamically on demand. Once a component is no longer needed it can be unloaded on the fly, allowing only necessary components to be loaded in memory at one time. The DCL (Dynamic Component Loader) is responsible for on-demand loading of needed components. Section 3.5 describes the currently implemented components in Tiny UIUC. The APA-Client (Authentication and Privilege Attribute Client) is responsible for secure connections with the authentication server.

3.2 Security server

In the security server there is an AS (Authentication Server), which provides a single sign-on point for the distributed environment. The KDC (Key Distribution Center) manages the cryptographic keys. The PAS

(Privilege Attribute Server) provides information about the privileges and security attributes of users and user sponsors. This information is provided to principals in the form of a special data structure referred to as the PAC – Privilege Attribute Certificate. The PAC is signed, as a protection against tampering, using the private key of the PAS. Different PACs can be issued to different users and devices to identify them. The PAC can contain the role-names assigned to a particular principal.

3.3 Service side

The service side represents the smart devices that users try to access remotely. At this side, the SACM, GSS-API and DCL are present and function as their counterparts on the client side. The PVF (PAC Validation Facility) is the component responsible for checking the validity of PACs and detecting any tampering.

3.4 Overview of tiny SESAME protocol

To be able to access the services securely, users are authenticated to the system by the AS, using *strong authentication*, which is an enhanced form of the Kerberos authentication scheme. Authenticating the user is done through the usual log-on password-based approach. The APA-Client communicates securely with the security server to authenticate the user. Upon a successful authentication, the user sponsor obtains a PAC from the PAS. The PAC contains the user's roles, security attributes and the basic key for communication with the remote service. The PAC is signed using the private key of the PAS. When the user wishes to use remote services (such as accessing smart consumer devices), the user's PAC is sent using the SACM to the desired service. The data sent is transferred by an ordinary protocol, such as Java RMI or TCP/IP protocols, to the service machine. The service passes the data received, via its SACM, to the PAC Validation Facility (PVF), which verifies, from the integrity protection, that the PAC is genuine and a secure association is established over which the user can communicate with the remote smart device. However, if mutual authentication is desired, where a user may want to make sure he is contacting the intended device, the PAC of the target device is sent back before the establishment of the secure association. Once all PACs are validated successfully,

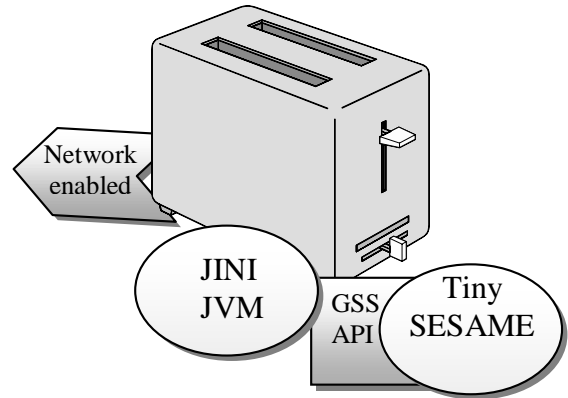


Figure 2: A Smart Device

the secure association can take place.

3.5 Component-based architecture

Mobile devices differ greatly in their capabilities, processing powers and security needs. What is needed is a security model that can adapt to the particular capabilities and security requirements of a smart device. Tiny SESAME achieves this by incorporating a component-based design for the client and service sides. In this design, the different security services, and protocols are implemented as separate components

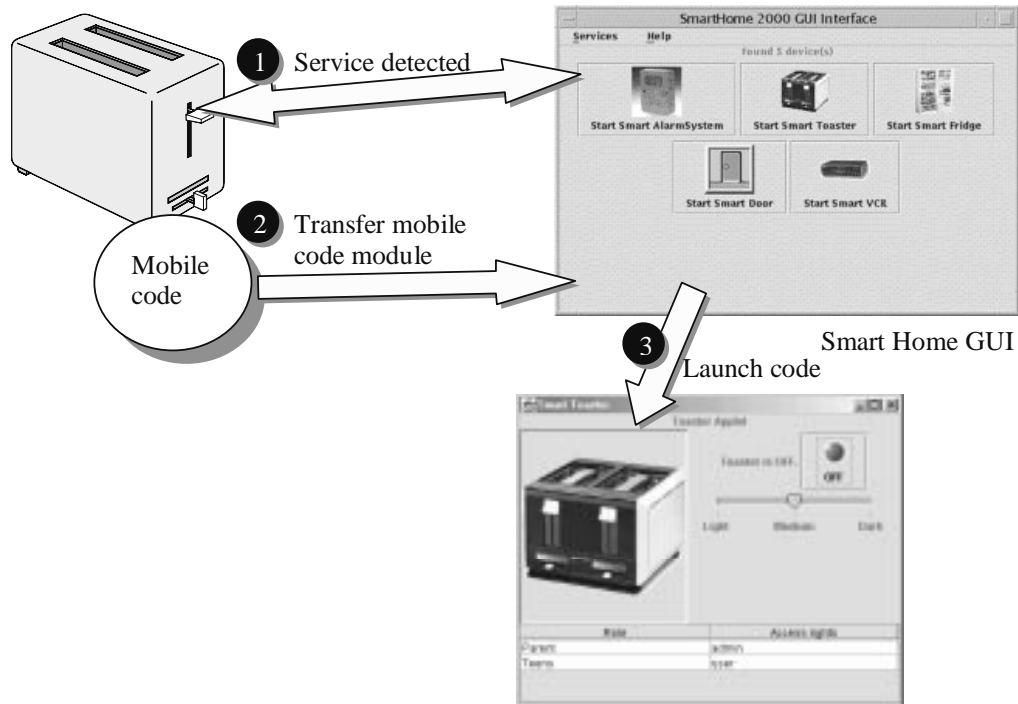


Figure 3: How the Smart Home System works

that can be loaded on demand. Thus, the security requirements of a device determine which components are loaded, effectively keeping Tiny SESAME's size manageable. As an example, consider a smart "light bulb". It may be desirable to have some minimal access control to prevent outsiders from tampering with the house's lighting, however it does not make sense to load an encryption library into such a small device.

Tiny SESAME consists of a core component that contains the essential functionality. This includes the Tiny SESAME factory object, the DCL and the GSS-interface without the actual implementation of the security services. The DCL is capable of loading other Tiny SESAME components from the device's built-in memory, on demand. The Tiny SESAME package contains several components. Currently, these include the SACM with low-encryption component, the SACM with high-encryption, the APA-client component, and the PVF component (Figure 1). For security reasons, Tiny SESAME components cannot be loaded from a remote source, rather a component must be loaded from local memory, for example, a device's built-in memory.

3.6 Size requirements of Tiny SESAME

The current implementation of UIUC SESAME [Cha99] is more than 1,240 KB of Java bytecode. The sizes of the current version of Tiny SESAME's components are listed in Table 1. Tiny SESAME's high-encryption component implements most popular

cryptographic functions, including symmetric, asymmetric, stream, and block encryption methods. The low-encryption component only provides a simple implementation of RC5 [Riv94]. We are working on providing more encryption components, including an elliptical-encryption component.

Table 1: Component sizes

Component	Approx. Size
Core Component	31 KB
SACM with high-encryption	371 KB
SACM with low-encryption (RC5)	59 KB
APA-Client	10 KB
PVF	7 KB

4. The nature of "Smart" Devices

We envision a Smart Home as an active environment populated with smart, dynamically configurable consumer devices that are capable of interacting with humans and other smart devices. We assume that the Smart Home is equipped with a Family-Area Network (FAN). We refer to the devices that support our system as "Smart Home devices". We assume that these household devices have a Java Virtual Machine (JVM) embedded within and are able to communicate with the surrounding environment using the Jini

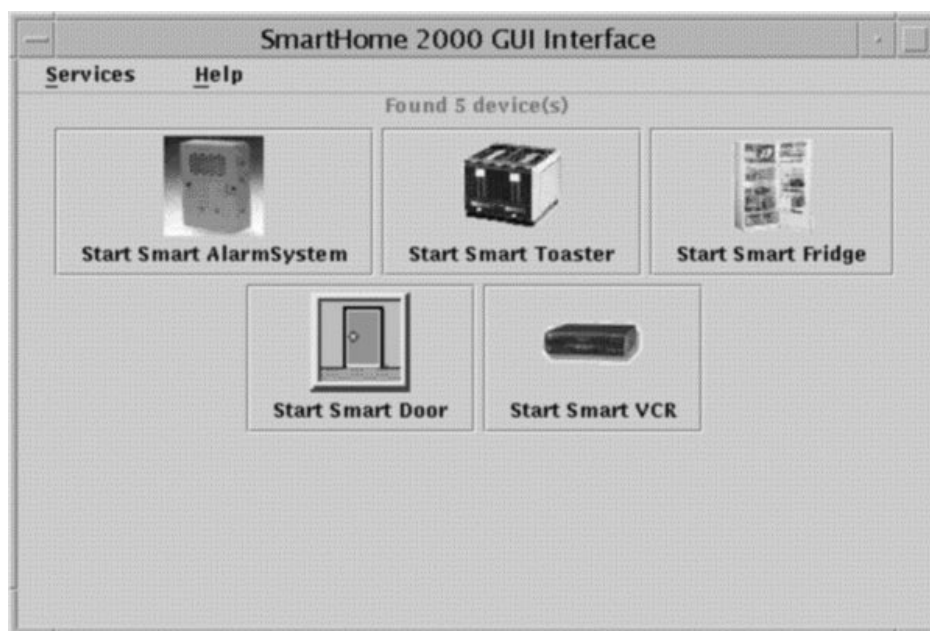


Figure 4: Smart Home client GUI, displaying the currently running devices.

protocols. Further, each device is network-ready and can be easily plugged into the nearest network socket.

To enforce security in the Smart Home environment, Smart Home devices will have an embedded version of *Tiny SESAME*. Tiny Sesame provides authentication, confidentiality, integrity, and access control services for smart devices.

4.1. Mobile code and dynamic interface

Since household devices vary significantly in their capabilities and functionality, each device may have a different interface for governing its configuration. For instance a “door” should provide an interface to open/close/lock the door. But for a VCR the interface should include controls for playback, rewind, eject, etc. Additionally, we would like our devices to be *truly* plug and play. Which means when a new smart device is employed, the consumer need only attach it to the network for the device to be instantaneously detected by the Smart Home GUI *without* the need for loading any device drivers.

To be able to achieve the above goals, each smart device must implement some “mobile code module” that extends Java’s “JApplet” class. Here, the smart device programmer can identify what functionality the end user can control and what security services to invoke. Once the GUI detects the device, the mobile code is transferred over the network using Jini protocols and is executed on the GUI’s location whenever the user wishes to configure that particular device. The GUI is capable of running and detecting new smart consumer devices without the need to add any drivers or interfaces to it. Figure 3 illustrates this.

It should be noted that the mobile code contains the GUI layout necessary for configuring the device in the form of Java Swing (or AWT) calls. However, the actual implementation of Swing is not transmitted in the mobile code, making the size of the mobile code manageable. That is, the mobile code invokes the GUI and security libraries at the host. The mobile code for a sample smart toaster device with several configuration options, an authentication dialog, and an access control configuration dialog is about 10 KB in size.

4.2. The Smart Home client

The Smart Home client (Figure 4) provides a GUI interface that can be used to configure all Smart Home compliant devices. It is able to detect the devices on the home network dynamically, without loading any special device drivers. It also displays icons representing the different devices. Since the smart home environment relies on Java, the homeowner is capable of running the client and configuring his smart devices anywhere, anytime using a PC, PDA, cellular phone or any device capable of connecting to the Internet and running a web browser. Upon clicking on the icon of a particular device, the client GUI will download the mobile code and run it enabling the user to configure the device-specific settings.

4.3 Security issues in mobile code

Mobile code provides greater flexibility and better utilization of distributed resources. By relying on Java, this allows the smart home environment to

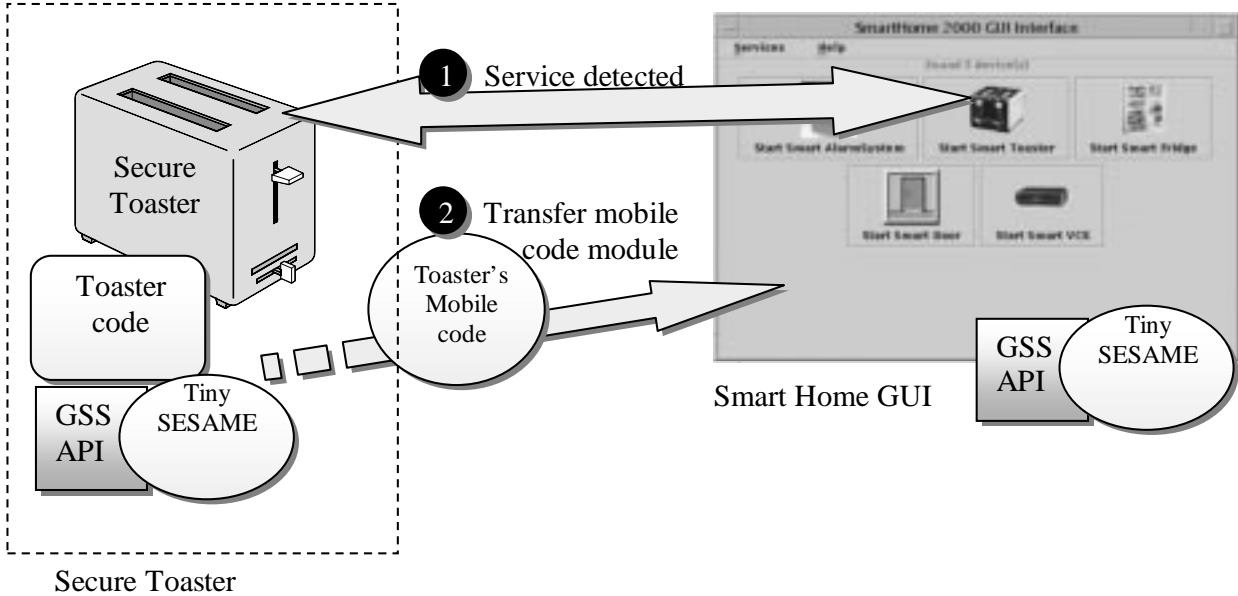


Figure 5(a): Integrating Tiny SESAME

transmit mobile code in the form of Java bytecodes that are capable of running on any client machine with a browser. Certainly, such flexibility has its price. Often, mobile code can lead to security hazards, particularly, when code is generated at untrusted sources. To counter the potential security threats that accompany mobile code, we employ Java's tactics: sandboxes and bytecode verifications [Fri96]. If resources permit, mutual authentication can be used to ensure that communication is established with the intended device. Code signing can be employed to detect tampering.

5. Integrating Tiny SESAME with Smart Homes

First of all, the security server (Figure 1) must be running somewhere within the FAN, for example, the home's central computer system. Figure 5(a) shows how Tiny SESAME is used in our Smart Home simulation. Each Smart Home compliant device has a

local copy of Tiny SESAME that is used by the device to validate the privilege attribute certificates (PACs) and to do data encryption and decryption, if necessary. As discussed earlier, each smart device has some mobile code. Once a user chooses to configure a particular device from the GUI, the mobile code is transferred to the GUI and launched. If the device requires authentication or enforces some access control, the mobile code uses the local copy of Tiny SESAME at the GUI client to authenticate the user (Figure 5(b)). If the authentication is successful, a PAC is generated and a security association is initiated between the mobile code and the remote smart device. The GSS-API provides calls for establishing a security association with whatever security features and QoP (Quality of Protection) needed by the service. The PAC is validated at the PVF component within the Tiny SESAME that is embedded in the smart device. This process ensures that the client is authorized to access the service. Upon establishing a security association successfully, data or commands can be

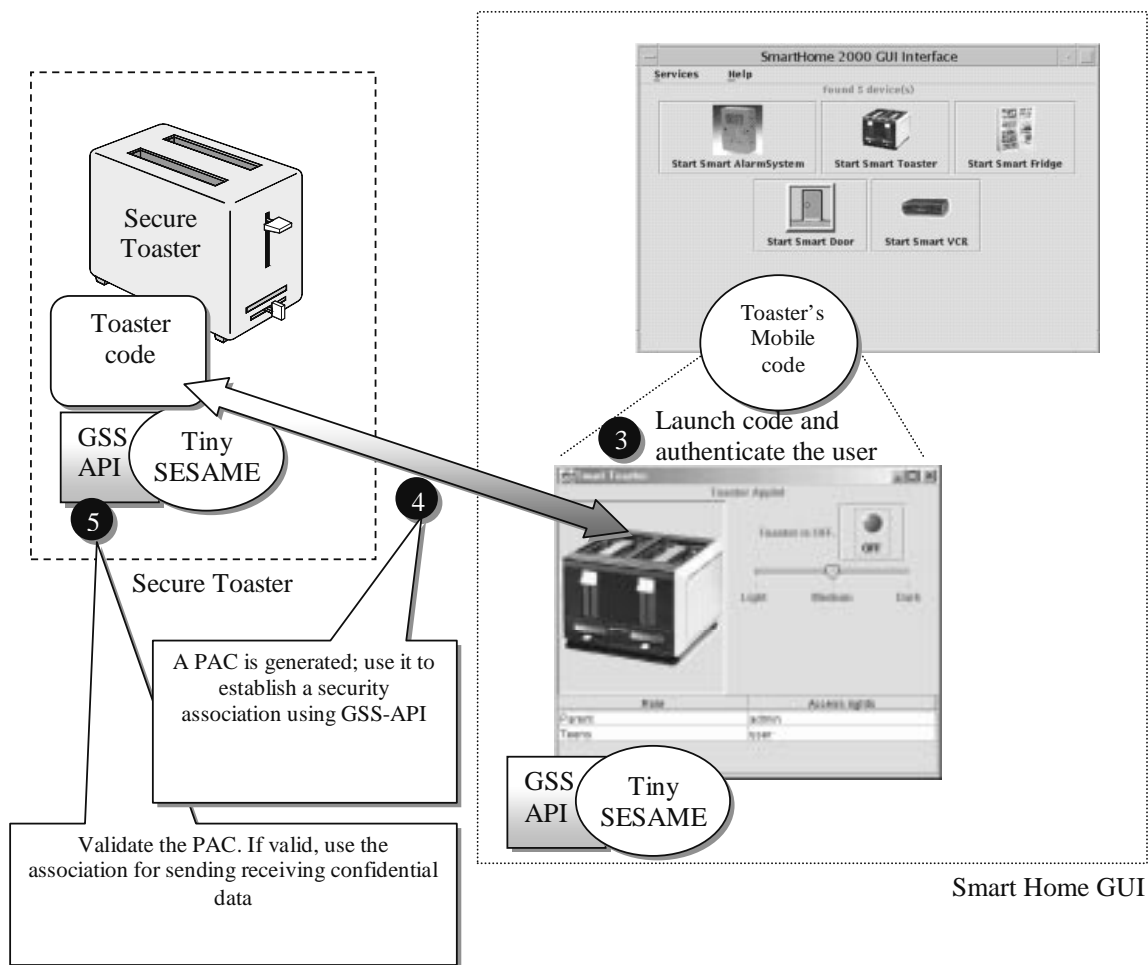


Figure 5(b): Integrating Tiny SESAME

sent over the association securely. As the PAC contains the role-names associated with the remote user, the device can grant access to the user according to his or her privileges.

As the case with new workstations, a newly manufactured smart device can be shipped with a default policy. When invoking the device for the first time, the administrator is asked to change the default policy and assign a valid PAC to the new device.

6. Conclusion

As more and more devices become smaller and mobile, and as smart environments and active spaces continue to bloom and attract small consumer and mobile computing devices, the demand for robust security services grows. Due to mobility, there is a limit to the processing power, energy and memory that a security mechanism can utilize. What is needed is a lightweight security component that can adapt to environments with scarce resources. Yet, this component should be able to evolve and extend once more resources are available. Tiny SESAME is such a component that can be easily ported to mobile devices while providing a subset of SESAME's security services. The combination of Tiny SESAME and Jini can be used to create a dynamic, secure environment of distributed computing devices.

7. Future work

We plan to incorporate some simple delegation capabilities in Tiny SESAME. We also plan to add additional encryption components and support for PKI. Some work is underway to port Tiny SESAME to Windows CE and PalmOS devices.

8. References

- [Ash99] P. Ashley, M. Vandenwauver, "*Practical Intranet Security: Overview of the State of the Art and Available Technologies*", Kluwer Academic Publishers, 1999.
- [Cha99] M. Chandak, "*U-SESAME: Achieving a Portable Authentication, Access Control, and Delegation Protocol*", Masters Thesis at Department of Computer Science, University of Illinois at Urbana-Champaign, 1999.
- [Ecm96a] ECMA-219, "*Authentication and Privilege Attribute Application with related key distribution functions*", 2nd edition, European Computer Manufacturers Association, March 1996, <http://www.ecma.ch>
- [Ecm96b] ECMA-235, "*Security in Open Systems – The ECMA GSS-API Mechanism*", European Computer Manufacturers Association, March 1996, <http://www.ecma.ch>
- [Edw99] W. Keith Edwards, "*Core JINI*", Prentice Hall PTR, 1999.
- [Fri96] J. Fritzinger and M. Mueller, "*Java Security whitepaper*", Sun Microsystems, 1996. <http://www.javasoft.com/security/whitepaper.ps>
- [Kai94] P. Kaijser, T. Parker, and D. Pinkas, "*SESAME: The Solution to Security for Open Distributed Systems*", Computer Communications, 17(7):501-518, July '94
- [Kai98] P. Kaijser, "*A review of SESAME Development*", Proceedings of the 3rd ACISP Conference, pages 1-8, 1998.
- [Ker] MIT's Kerberos Homepage, <http://web.mit.edu/kerberos/www/index.html>
- [Lin93] J. Linn, "*Generic Security Service Application Program Interface*", September 1993, RFC 1508
- [Lin94] J. Linn, "*Generic Interface to Security Services*", Computer Communications, 17(7), July 1994
- [Lin97] J. Linn, "*Generic Security Service Application Program Interface Version 2*", January 1997, RFC 2078.
- [Mcm95] P. McMahon, "*SESAME V2 Public Key and Authorisation Extensions to Kerberos*", in proceedings of the symposium on Network and Distributed System Security, pages 114-131, February 1995.
- [Muf94] S. Muftic and M. Sloman, "*Security Architecture for Distributed Systems*", Computer Communications, 17(7) pages 492-500, July 1994.
- [Muh99] J. Al-Muhtadi, "*SGSS-API Documentation and Developer's Guide*", 1999.
- [Neu94] B. Neumann and T. Ts'o, "*Kerberos: An Authentication Service for Computer Networks*", IEEE Communications Magazine, 32(9): 33-38, September 1994

- [Riv94] R. Rivest, “*The RC5 Encryption Algorithm*”, Proceedings of the 2nd International Workshop on Fast Software Encryption, pages 86-96, Belgium, December 1994.
- [Ses] SESAME. The SESAME homepage <https://www.cosic.esat.kuleuven.ac.be/sesame/>