ABTTP: A TTP-based Prototype for Protecting Confidentiality of Sensitive Data with Active Bundles

Lotfi Ben Othmane, Leszek Lilien, *Senior Member IEEE*, Bharat Bhargava, *Fellow IEEE*, Mark Linderman, *Senior Member IEEE,* Rohit Ranchal, and Raed M. Salih

Abstract— The main challenges in information sharing are limitations of mechanisms for protecting confidentiality of sensitive data. An owner of the data may not be able to enumerate all entities that are allowed to access his data. The common approach to solve this problem is to attach privacy policies to the data. This approach assumes that the recipient's hosts enforce the policies attached to the data. A solution that relaxes this assumption is to use active bundles which are containers with a payload of sensitive data, metadata, and a virtual machine (VM) specific to the active bundle. This paper investigates the question: Can data protect their own confidentiality? To answer this question we developed the ABTTP prototype. We assume trustworthy execution of VMs included in active bundles by requiring that hosts excuting VMs are Trusted Platform Modules enabled. Our ABTTP implementation uses a mobile agent framework. The prototype protects privacy of sensitive data through: (i) assuring enforcement of privacy policy by using VMs, (ii) using host trustworthiness to activate protection mechanisms when data is tampered with, and (iii) recording all data-related activities by its VM. The prototype demonstrates the solution in a mobile agent environment. It proves that data can protect its own confidentiality.

Index Terms—active bundle, data dissemination, JADE, mobile agent, privacy, security, sensitive data, trusted third party.

1 Introduction

Data confidentially concerns exist whenever sensitive data are created, disseminated, stored, or used. Unauthorized data disclosures are among the main data confidentiality threats. The challenge in protecting sensitive data is to satisfy the conflicting needs of disseminating data while protecting them from unauthorized disclosures [15].

This introduction is organized as follows. Subsection 1.1 defines the basic terms used in this paper. Subsection 1.2 provides its motivation. Subsection 1.3 states the research problem and the contributions of the paper. Subsection 1.4 outlines the organization of the paper.

1.1. Definitions of Basic Terms

Sensitive data (a.k.a. *private data*) are data whose owners do not want made public. (A *data owner* is typically the data creator or originator.)

Data sensitivity factors, which make data sensitive, include the following: (cf. [21]): (i) being inherently sensitive—e.g., location of defense missiles; (ii) coming from a sensitive source—the source implies that the data are sensitive; (iii) being declared sensitive by the owner; (iv) being in a sensitive context—e.g., an individual's address in the context of a court trial; (v) being a part of sensitive data—e.g., "Chicago" could be a part of a sensitive address data "1904 S. Michigan Avenue, Chicago"; (vi) being sensitive in relation to previously disclosed data.

Data lifecycle (cf. [3]) includes: (i) creating data; (ii) disseminating data from its creator to one or more hosts (including storing data after receiving them, and retrieving data before forwarding them); (iii) disseminating data among hosts; (iv) reduction of data value through aging or a voluntary *evaporation* (a partial self-destruction of data) when threatened with disclosures [19]; and (v) *apoptosis*—a total self-destruction of data.

Data dissemination is the process of transfer (i.e., forwarding) of data from a souce entity (e.g., a human, a host, software, etc.) to a destination entity (cf., [3]). A *guardian* is an entity (either human or not) that accesses data or disseminates it. The owner of sensitive data transfers them to a set of guardians. In turn, a guardian may transfer the received sensitive data to another set of guardians.

We distinguish two *data dissemination styles* (cf. [23]): (1) *data push,* when data is sent to a recipient by a guardian; and (2) *data pull,* when each recipient obtains the data from a data server (an external repository), where data were previously stored by guardians.

Confidentiality of data means ensuring that data are not made available or disclosed to unauthorized human or artificial entities [27].

One of the basic components used by mechanisms for protecting data confidentiality is the *privacy policy*. Privacy polices specify a set of *data disclosure rules* that govern the operations on given data (such as the read, write, or delete operations).[1] As an example, a privacy rule can

- *Lotfi Ben Othmane is with the Department of Mathematics & Computer Science, Eindhioven University of Technology, Eindhoven, The Netherlands. E-mail: l.ben.othmane@tue.nl.*
- *Leszek Lilien and Raed Salih are with the Department of Computer Science, Western Michigan University, Kalamazoo, MI 49006. E-mails:{leszek.lilien, raed.salih}@wmich.edu.*
- *Bharat Bhargava and Rohit Ranchal are with the Department of Computer Science, Purdue University, West Lafayette, IN 47907, Emails: {bbshail, rranchal}@purdue.edu.*
- *Mark Linderman is with the Air Force Research Lab, Rome, NY 13441, email: mark.linderman@rl.af.mil*

Manuscript received (…).

[1] These policies and rules should be called *confidentiality policies* and *confidentiality rules* but a historical precedent outweighs naming precision.

specify conditions for data access, e.g.: "The marketing department is allowed to read contact data for the purpose of e-mail marketing." A privacy can also specify obligations, e.g.: "Delete contact data of the marketing department after 30 days" (cf. [15]).

Privacy policies are verified by evaluating privacy rules, which often require checking a set of credentials. The evaluation output (resulting from the policy) can be used by an enforcement mechanism deciding whether a requested operation is authorized by the data owner or not.

Privacy policies are usually specified using *policy languages*, such as OASIS eXtensible Access Control Markup Language (XACML) [20]. Another method to specify privacy policies is to use *Role-Based Access Control (RBAC)* [15]. RBAC models assign users' access permissions for resources based on users' roles.

A privacy-enabled system, such as E-P3P [15], includes a set of components ensuring that the usage of data in an enterprise complies with the privacy policy.

1.2. Motivation for Protecting Confidentiality of Sensitive Data

A sensitive data owner could be willing to share her sensitive data with a set of entities. In many cases, the owner does not know *all* entities that may be allowed to access data, hence does not know the public keys of *all* entities that could access data. However, she could specify a set of criteria that are satisfied only by the entities that may access the sensitive data. These criteria could be specified in terms of privacy policies. The owner may also want to specify a set of obligations for the sensitive data disclosure.

The active bundle solution proposed by us in this paper can be used by a number of applications, including the following ones:

1. *Pervasive Health Monitoring Systems and Devices:* Use of pervasive medical devices is growing. The devices (ofter worn by or implanted in patients) generate healthcare data that are disseminated and accessed by many healthcare providers. The availability of this information to a healthcare provider could be critical for saving the life of a patient. However, allowing even inadvertent dissemination of this information to a party that may be unauthorized (such as a news media) may be damaging to the patient or caregivers.
   Note that some current models of pervasive medical devices (such as insulin regulation devices and blood measurement devices) use the mobile agent [11].
2. *Smart Grid:* The Smart Grid allows energy consumers to buy energy, store energy, and sell it [30]. The distribution of energy is based on the information collected from the different partners (including customers). Unauthorized modification of information exchanged by the parties could disrupt the distribution of energy to customers.
3. *Digital Rights Management*: Digital rights management blocks access to digital content (e.g., documents, movies, etc.) by entities not authorized by the content providers [32] who sell their digital contents. Unauthorized use of digital content would be a loss for them.

1.3. Research Problem and Paper Contributions

This paper addresses the question: *Can data protect their own confidentiality?*

We answer this question positively. We describe the ABTTP, our prototype implementing the active bundle scheme using a trusted third party (TTP). Our solution does not require the existence of a client application at the visited host that enforces a privacy policy associated with the sensitive data. (Solutions that require such clients typically also assume that the clients are trusted by the owner of the sensitive data.)

Much of research on protecting data uses a TTP to perform computations on data from untrusted parties. In contrast, we use a TTP to store decryption keys and deliver them only to hosts that have a sufficient trust level for accessing all or part of the data.

In this paper we do not consider the authenticity or legitimacy of sensitive data, and do not deal with piracy against sensitive data. Possible solutions for data authentication could use, for example, digital certificates for data. Solutions against piracy could be based, for example, on watermarking and forensic approaches.

1.4. Paper Organization

The paper is organized as follows. Section 2 describes related work. Section 3 gives an overview of mobile agents. Section 4 describes the active bundles approach and design. Section 5 provides the threat model. Section 6 describes the ABTTP in the mobile agent architecture. Section 7 describes the configuration setup and two experiments with the ABTTP. Section 8 concludes the paper.

2 Related Work

Various solutions for protecting confidentiality of sensitive data are proposed in the literature. Park, Sandhu, and Schifalacqua [23] present a taxonomy of architectures for managing access control and dissemination control for sensitive data and digital content. The classification is based on using or not using metadata for access and dissemination control, using or not using virtual machines (VMs) located on the hosts, and the dissemination style.

The authors note that using a host's VM reduces security of digital content. Solutions using a VM at the receiving host assume that VMs at all hosts are fully trusted. This assumption is too strong to be generally acceptable. In contrast, our active bundle scheme requires only that an active bundle itself, not each host, includes a trusted VM.

Several solutions proposed in the literature assume that entities receiving data are known a priori. The owner can use an encryption mechanism to encrypt data with keys in such a way that each entity can decrypt data that it is allowed to access by confidentiality restrictions (e.g., policies). Digibox [25] is an example of a solution for protecting confidentiality of sensitive data using multiple keys. Digibox is a cryptographically protected container with data and controls that enforce rights to these data.

Other solutions proposed in the literature address the problem of protecting confidentiality of sensitive data where entities receiving data are not known a priori (which is the problem that we address in this paper). We describe in the following three solutions for protecting confidentiality of sensitive data.

2.1. Protecting Confidentiality of Sensitive Data Using Sticky Policies and Identity-based Encryption

A Trusted Platform Module (TPM) is a security hardware component. It can facilitate numerous important security capabilities, such as authentication, hardware-based encryption, digital signing, secure key storage, and attestation of software installed on hardware. For encryption and signing, a TPM uses keys stored in it and are not accessible (only used in the TPM) [28].

A TPM enables building trustworthy computing relationships. For instance, it can be used to prevent tampering software through exploiting the chain of trust [22].

Montero *et al.* [13] propose a technique for enforcing confidentiality protection during dissemination of sensitive data (such as digital identities) using sticky policies. Sticky policies are policies that are inseparable from the associated data. Montero *et al.* investigate how to implement sticky policies; that is, how to make data and their policies inseparable in a way that an attacker cannot access data without satisfying their privacy policies. They identify two approaches for associating data with their related policies: (i) use of identifier-based encryption, and (ii) use of a (TPM).

In the first approach, data is sent from their owner to a receiver who may disseminate them further. The owner constructs the encryption key using the receiver's identifier, constraints, and conditions defined by the data access policies. The owner also encrypts data, and sends data and their associated data access policies to a receiver. The receiver provides its credentials and data access policies to a trusted authority (TA). TA computes the decryption key using the receiver's credentials and the sticky policies. Then, the TA sends the key to the receiver. The decryption key can decrypt data only if the receiver provided the TA with the appropriate credentials for the sticky policies associated with these data.

The second approach, using a TPM, enforces the data access policies associated with a given data set. This approach can be compromized in at least two ways:

1. If data and policies are received by a host that has no TPM *and* no policy engine, data cannot be accessed. In contrast, in our scheme we do not require hosts to have their own policy enforcement engines; instead, it is a part of the bundle's VM.
2. A policy enforcement engine available at a host could be changed by the host in such a way that the host can access data against the "will" of the data access policies. In contrast, in our scheme hosts do not own policy enforcement engines; instead, the privacy policy engine is a part of the bundle's VM. (Our privacy policies include data access policies).

A different solution for a policy-based cryptographic scheme was developed by Bagga and Molva [5] which uses an identity-based encryption scheme. The scheme allows: (i) performing data encryption operations using privacy policies formalized as Boolean expressions, and (ii) performing data decryption using policies and digital credentials of an entity that wants to decrypt these data.

2.2. Protecting Confidentiality of Sensitive Data Using Activation Time

Casassa Mont *et al.* [7] propose a solution for document dissemination. A document owner plans to send it to a set of recipients. He generates a symmetric key *SK*. Then, encrypts the document using a *SK*. Next, he encrypts the *SK* using an encryption key based on the planned date and time of the disclosure of the document (e.g., "GMT201009101200" is an encryption key for documents to be disclosed at 12:00 a.m. on Sep. 10, 2010). Next, the encrypted document is distributed to intended receivers.

The solution uses a *time vault service*, which continually generates and publishes decryption keys associated with the current date and time (obtained by a trusted clock).

A receiver can open a document only with a decryption key, provided to the document receiver not with the document but later, at the document disclosure time. The receiver gets this key from the time vault service. Then, the receiver uses the decryption key (obtained from the time vault) to decrypt the symmetric key. Next, the receiver decrypts the document using the symmetric key.

In contrast, in the active bundle scheme the rules to disclose data are not limited to using timing conditions only. Instead, data from an active bundle are disclosed to any receiver that satisfies the privacy policies included in the bundle.

2.3. Protecting Confidentiality of Sensitive Data Using Bundles and Active Bundles

Lilien and Bhargava [18] present an approach for protecting disseminated sensitive data using the bundle scheme. A *bundle* packages sensitive data and metadata associated with these sensitive data. Metadata includes policies for data access and dissemination. The scheme assumes that hosts receiving bundles can be trusted to enforce privacy policies defined in the bundle's metadata.

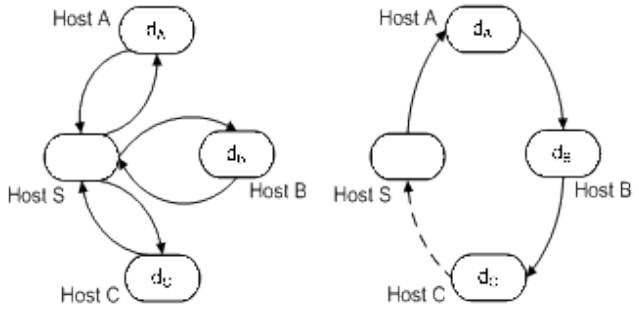
Active bundles remove the trusted host assumption, and associate a VM with the bundle, making it "active." The VM enforces privacy policies, and uses several protection mechanisms.

3 Overview of Mobile Agents

A *mobile agent* (a.k.a. a *mobile object*) is a software object able to perform computations on visited hosts, transport itself from one host to another, and interact with and use capabilities of visited hosts [17]. Being a piece of software,

a mobile agent is "passive" and it is "activated" by having its code executed by its host. A mobile agent contains code and carried data, which is data that it brought with it to the host that it is currently visiting.

A mobile agent is created by an *originator* and *disseminated* to one or more hosts. A *receiving host* may disseminate the mobile agent further to other hosts. Alternatively, a mobile agent can *self-disseminate* to one or more hosts.



a) Client-server architecture b) Mobile-agent architecture

Fig. 1. Client-server architecture vs. mobile-agent architecture

The most common architecture for applications using distributed data is the client-server architecture. An alternative is to use mobile agent architecture. Fig. 1 illustrates the major difference between these two architectures. In this example, an application that starts on Host *S* needs data distributed over three hosts: Hosts *A, B*, and *C* with data d_A, d_B, and d_C, respectively. In the first, client-server architecture, data must be sent to *S* from *A, B*, and *C* as shown in Fig 1a. In contrast, in the second architecture, a mobile agent visits *A, B*, and *C* to access their data d_A, d_B, and d_C, respectively, as shown in Fig. 1b (the broken line indicates that the mobile agent may—but does not have to—return to its original host *S*). The cost of executing a mobile agent is spread over *A, B*, and *C* rather than being concentrated on *S*, as in the client-server architecture.

Despite being a concept known for years, mobile agents are still a very active research area [10, 24]. In a large measure this is due to advantages offered by them to important new paradigms or technologies.

Some of the advantages are: reducing network load, overcoming network latency, encapsulating protocols, executing asynchronously and autonomously, dynamic adaptability, robustness and fault tolerance [17]. However, the mobile agent paradigm has a limited security [19].

4 Active Bundle Approach and Design

We discuss in this section our approach for protecting sensitive data using active bundles and the design of the active bundle scheme.

4.1. Active Bundles Approach

In current confidentiality (and security) solutions, data are commonly considered as passive entities that cannot protect themselves. As a consequence, they require the use of another active and trusted entity to protect them (e.g., a trusted processor, a trusted memory module, or a trusted third party). We propose an approach that transforms passive data into an active entity by encapsulating them within active bundles by becoming an inextricable part of an active bundle.

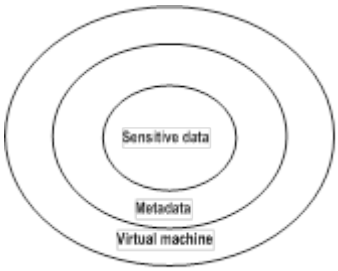


Fig. 2. The basic structure of an active bundle

The *active bundle scheme* [3] is based on a software construct named an *active bundle*, which—as illustrated in Fig. 2—bundles together sensitive data, metadata, and a virtual machine specific to the bundle. *Metadata* contain information about data carried by the active bundle. They specify policies for dissemination of data by active bundles (the policies include access control rules for these data). VM[2] controls its active bundle and enforces the policies specified by metadata. Sensitive data, metadata, and the VM of an active bundle are collectively called a *payload* of the bundle.

The active bundle scheme protects confidentiality of sensitive data by: (i) enforcing privacy policies specified within a bundle's metadata, (ii) activating protection mechanisms when data are tampered with, and (iii) recording activities involving a bundle (e.g., a transfer of the bundle from a source to a destination).

The active bundle scheme protects sensitive data as long as they are accessed through an active bundle. Sensitive data or their parts are not protected once they are given as an output to any untrusted entity (since they leave the sphere of control of the scheme).

4.2. Design of Active Bundle Scheme

We discuss in this subsection the structure and operation of active bundles.

4.2.1. Structure of an Active Bundle

Fig. 2 shows the structure of an active bundle [3]. An active bundle includes the following components:

1. *Sensitive data*: It is a digital content that needs to be protected from privacy violations, data leaks, unauthorized dissemination, etc. The digital content can include documents, pieces of code, images, audio, or video files.
2. *Metadata*: It describes the active bundle and its privacy policies. It includes (but is not limited to) the following components:
   a. *Provenance metadata*—including an identifier of the active bundle, identifiers of its creator and owner, the creation date, identifiers of all visited hosts, as well as identifiers of all guardians with their access timestamps and, possibly, their update timestamps if they performed any updates of sensitive data. The history of visits and updates is kept private (it

[2] In the context of active bundle we use a specific virtual machine that has specific functions. It is different from the commonly known virtual machine such as VMware which runs an operating system.

could be used only for forensic investigations by authorities that obtain a court order).

b. *Integrity check metadata*: It includes the algorithm for checking integrity of sensitive data as well as a hash calculated by the owner.

c. *Privacy policy metadata*: It includes access control policies for sensitive data of the bundle. They could use the required minimal host's trust level for each subset of sensitive data.

d. *Dissemination control metadata*: It includes dissemination policies specifying who can disseminate the active bundle, and under what conditions (e.g., the active bundle could be disseminated at a specified time, or could be disseminated to a specific group of addressees—such as employees, or a board of directors.)

e. *Life duration*: It specifies a date and time where the sensitive data must disappear.

f. *Security metadata*: It includes: (i) *security server id*, specifying the security server used in the process of bundle encryption and decryption; (ii) *encryption algorithm* used by the VM of the bundle; (iii) *trust server id*, used to validate the trust level and the role of a host at which the bundle arrived; and (iv) *trust level threshold*, specifying the minimal trust level required by the VM to enable the active bundle.[3]

g. *Other application-dependent and context-dependent components*: It includes information related to semantics (the application, the context, etc.) of sensitive data of the active bundle.

3. *Virtual machine (VM)*: The role of a VM is to guarantee the appropriate access control to sensitive data of the bundle (for example, disclosing to a guardian only the portion of sensitive data that the guardian is entitled to access). A VM also performs three control mechanisms operations that we describe in ubsection 4.2.2.

The main challenge of implementing the VM is how to assure that a visited host executes the VM code faithfully and correctly. We believe that using a TPM is the best solution to solve this issue. We assume that an operating system (OS) certified by a TPM executes correctly a VM code. Using a trusted VM is another possible solution. Even if attackers change the VM code, integrity self-checks will detect such tampering [8]. Hence, the OS will execute the correct VM code.

4.2.2. Operations of Active Bundles

The three operations performed by an active bundle are:

1. *Evaporation:* After arriving at a host, an active bundle asks for the host's trust level. If the hosts' trust level is sufficient to allow access to all or portion of the bundle's data, then the bundle's privacy policy is applied. All data that the host is not allowed to access (as specified in the privacy policy) might be "evaporated." Evaporation[4] of data diminishes the value of data.[5]
2. *Apoptosis:* An active bundle may realize that its security or privacy is about to be compromised, e.g., it may discover that its self-integrity check fails, or the trust level of its guardian is too low. In response, the bundle may choose to apoptosize (i.e., perform a clean self-destruction; that is, a complete self-destruction that leaves no trace usable for an attacker [18, 29]). Rules for triggering an apoptosis are included in the privacy policies of bundle's metadata.
3. *Integrity self-check:* Upon activation of an active bundle at a new guardian, the bundle checks its integrity using the algorithm specified in its metadata. It calculates the hash value of the active bundle, and compares the computed value to the value recorded within the bundle metadata. If the values differ, the bundle performs apoptosis.

An active bundle might perform other operations, which are beyond the scope of this work. For example, an active bundle may perform *decoy operations* by spreading misleading information at the visited hosts (even if it is not enabled at them) to confuse potential attackers.

5 Threat Model

Sensitive data can be compromised in different ways. ABTTP protects sensitive data only from the following set of *confidentiality threats*: unauthorized data disclosures, unauthorized data modifications, unauthorized data disseminations. Other threats, which are security threats but *not* confidentiality threats, are beyond the scope of this paper. Examples of such threats include: data inferences, Sybil attacks, and masquerade attacks.

ABTTP ensures enforcement of confidentiality policies associated with sensitive data even on untrusted hosts. This is accomplished since untrusted hosts cannot obtain from the TTP decryption keys for protected data.

The solution relies on the following assumptions [4]:

A1. Access to a TTP that stores decryption keys and provides them to enabled active bundles is assured.

A2. The TTP can obtain from a *trust server* information indicating trustworthiness of any host visited by an active bundle. Host trustworthiness information can be obtained through different approaches, including remote attestation of hosts [9]. It is further assumed that the trust level of a visited host is indicated with a single numerical value.

A3. Any host receiving an active bundle has a TPM and correctly executes the code included in the VM of the bundle.

If a host receiving an active bundle disables execution of this code. We require that a trust evaluation server (TES) notifies the security server that a host is not trusted when TES evaluates that the host does not execute the bundle's code correctly. If the host stops execution of the code before apoptosis is activated, this information cannot be used to activate apoptosis (since the code that could activate apoptosis is already stopped). However, after receiving such notification, the security server denies the host's requests for decryption keys, which prevents the host from accessing the bundle's data.

A host that tampers with execution may disallow posing requests for decryption keys from the secu-

[3] If an active bundle is received by a destination host H with the trust level t, the VM of the bundle must ensure that H has access to only the portion of data for which t is a sufficient trust level. If only a portion of data is *not* to be seen by H, evaporation of the data *not* to be shown to H is performed by the VM. If no bundle data can be seen by H, apoptosis is triggered by the VM.

[4] The name "evaporation" has been chosen because the host's "temperature" determines the extent of data "evaporation."

[5] One way to achieve evaporation is by replacing the original, more valuable data with approximate data (summary data, averages for ranges, abstracts for reports, etc.), or replacing more current data with older data [18].

rity server by bundle's code. In such case it cannot access the sensitive data included in the active bundle.

A4. There is a secure communication channel between active bundles and TTPs.

A5. Security server and hosts receiving active bundles authenticate active bundles—possibly by using certificates for active bundles. The certificates are issued by other TTPs.

We also note the following features and limitations of ABTTP:

1. A malicious host can copy an active bundle before activating it. As long as the host does not obtain decryption keys from a TTP, it cannot access data stored in the ciphertext form within the bundle.
2. Trustworthiness of a host is dynamic; that is, it may change over time. The most dangerous situation occurs when a host authorized by a bundle's privacy policy to access some (or even all) its data becomes a host that is not authorized to access these data.
   If the host obtained decryption keys when it was still authorized, it can keep accessing bundle's data after it lost its authorization. If the host obtained data when it was still authorized, it can keep disseminating these data after it lost its authorization. (We see no solution for this situation: the genie is already out of the bottle). In both situations, the host is analogous to a person that breaks one's trust.
3. ABTTP does not protect from side-channel attacks. For example, a host may cheat by accessing the memory used by the bundle. It can get decryption keys, and use them to decrypt data that the associated policies do not authorize it to access.

6 Description of the ABTTP in the Mobile Agent Architecture

This section is organized as follows. Subsection 6.1 presents the design of active bundles as mobile agents. Subsection 6.2 presents the tools and framework used in the prototype. Subsection 6.3 describes the architecture of ABTTP. Subsection 6.4 describes the behavior of the ABTTP components. For our and readers' convenience, we often use the "AB" acronym to denote "Active bundles."

6.1. Design of Active Bundles as Mobile Agents

A mobile agent is composed of state (which is its data) and code. Fig. 3 shows the components of a mobile agent. An active bundle is composed of sensitive data, metadata, and a VM. These three components could be classified into state and code where sensitive data and metadata compose the state and a VM composes the code. Therefore, active bundles and mobile agents have a similar structure.

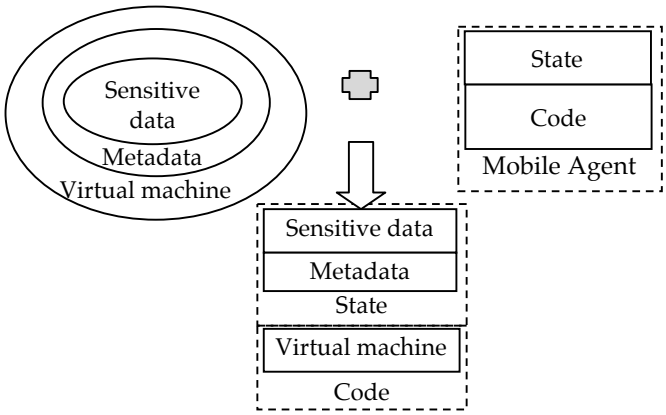


Fig. 3. Embedding an active bundle in a mobile agent

In this work we implement an active bundle as a mobile agent. To implement an active bundle as a mobile agent, we use the VM as the code of a mobile agent and we use the sensitive data and metadata as the data of the mobile agent. Fig. 3 shows how the components of an active bundle are embedded into the mobile agent structure to obtain the active bundle structure.

Active bundle approach benefits from the capabilities of the mobile agent architecture paradigm (e.g., using its dissemination style). The approach enables protecting confidentiality of mobile agents through: (1) embedding protections mechanisms of an active bundle into a mobile agent, (2) using a TTP for assuring that only trusted hosts can obtain decryption keys to access sensitive data included in the active bundle, and (3) assuring enforcement of privacy policies through the active bundle's VM.

6.2. Tools and Framework used in the Prototype

We have developed ABTTP in the context of the mobile agent framework JADE [2]. Other major tools used for implementing the prototype include the programming language Java 1.6.01 [14], the development environment Eclipse 3.5.2 [12], and Java cryptographic libraries [16]. JADE and Java cryptographic libraries are discussed next.

6.2.1. Mobile Agent Framework JADE

JADE is a software framework that provides basic mobile agent middleware functionalities independent of the specific application. These functionalities simplify realizations of distributed applications that are based on the mobile agent paradigm [2]. Fig. 4 illustrates the architectural elements of JADE. The *JADE framework* (i.e., libraries) enables the development of multi-agents applications. The *JADE platform* enables deployment of multi-agents applications on one or more hosts.

JADE is developed using Java. The JADE platform is composed of *agent containers* (or just *containers*) that can be distributed over network nodes. A *platform* is a distributed application that includes containers joining and leaving it dynamically. A *container* is a Java process that provides the JADE run-time services and all other services needed for hosting and executing agents. Containers can be distributed on a JADE platform among a set of hosts that communicate with each other. The *main container* is a special container that represents the bootstrap point of a JADE platform. It is the first container to be launched by a platform, and all other containers must register with the main container [2].

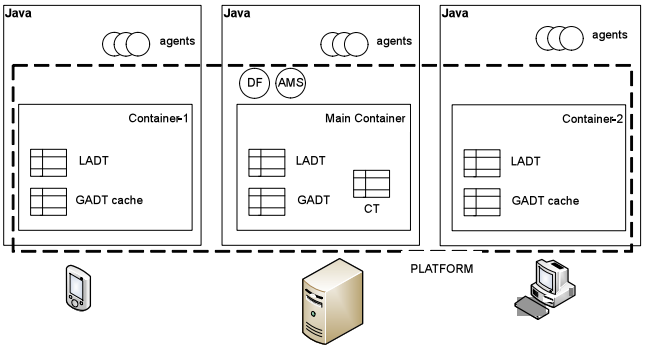


Fig. 4. Architectural elements of JADE (cf. [2]).

The main container provides a number of functionalities, including the following [2]:

1. *Hosting the Directory Facilitator (DF). DF* is a special agent that provides default yellow-page services for the platform. The yellow pages[6] service is used by agents searching for available service, or wishing to register their own services.
2. *Hosting the Agent Management System (AMS). AMS* is a special agent that supervises the entire platform; among others, it provides management of agents' lifecycles, and white-page services. AMS is the contact point for all agents that need to interact with each other.
3. *Managing the Local Agent Descriptor Table (LADT). LADT* is a registry of agents available in the local container. Management of LADT includes maintaining their current status.
4. *Managing the Global Agent Descriptor Table (GADT). GADT* is a registry of agents for a platform. Management of GADT includes maintaining their current status and locations. Each container has its own LADT, while there is only one GADT, which includes information from all LADTs. The GADT cache is a copy of GADT at a specific time (GADT is synchronized periodically with all LADTs).
5. *Managing the Container Table (CT). CT* is a registry of containers. Management of CT includes maintaining references and transport addresses for all containers for a given platform.

A mobile agent in JADE is a single-threaded Java program that has a set of behaviors. A *behavior* is a Java object that executes a single task, such as "Send this message," or a complex task in a thread. When the behavior object finishes its execution, it gives the control to its agent [2].

A mobile agent communicates with other agents using asynchronous message passing. Each agent has a mailbox where messages sent by other agents are posted, until they are picked up for processing by the agent. Messages are structured using ontologies for agent communications. (An *ontology* provides a description of concepts and relationships between them.) Mobile agents communicating with each other must know this ontology; otherwise they would be unable to encode and decode the messages they exchange [2].

6.2.2. Java Cryptographic Libraries

Java cryptographic libraries are composed of two components [16]:

1. *Java Cryptography Architecture* (*JCA*). JCA is a library included in the Java Development Kit (JDK). It *defines* cryptographic concepts; the descriptions (but not implementations) are encapsulated in classes that compose java.security and javax.crypto packages
2. *Java Cryptography Extension* (*JCE*). JCE is an extension library, including algorithms absent from JCA (and, thus, from JDK). JCE implements the descriptions defined in JCA in the namespace javax.crypto.

The libraries are used in the ABTTP prototype for: (i) encrypting and decrypting active bundles, (ii) generating encryption and signature keys, (iii) computing the cryptographic hash values for sensitive data included in active bundles, and (iv) signing active bundles.

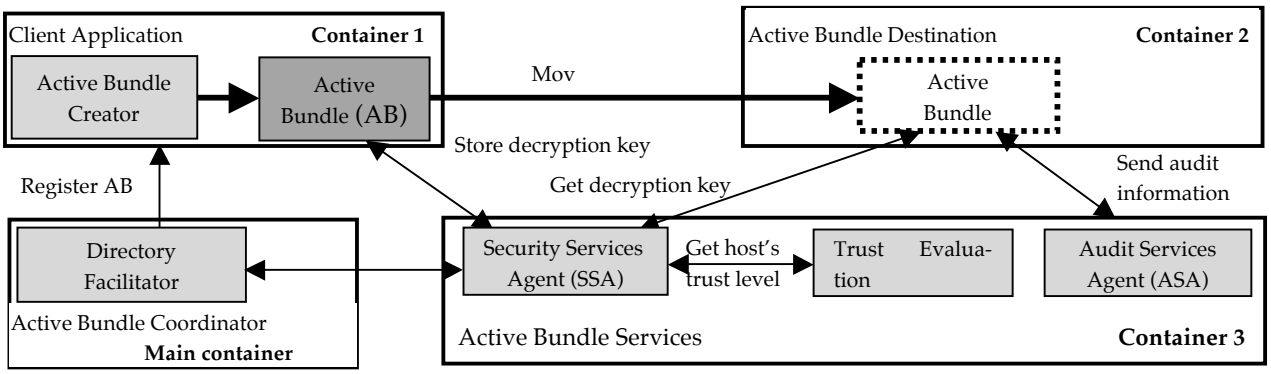


Fig. 5. UML component diagram [6] of the ABTTP prototype: a TTP-based active bundle implementation.

6.3 Description of the ABTTP Architecture

The components of our prototype, illustrated in Fig. 5, are distributed among four containers:

1. *Main Container—AB Coordinator* includes Directory Facilitator (DF).
   Container 1—Client Application includes AB Creator that creates active bundles; one active bundle is shown explicitly in the figure. (Note that an active bundle is a mobile agent, not a Java class).
2. *Container 2—AB Destination (ABD)* includes disseminated active bundles. One active bundle that arrives at ABD is shown explicitly in the figure.
3. *Container 3—AB Services* includes Security Services Agent (SSA), Trust Evaluation Agent (TEA), and Audit Services Agents (ASA). All three are TTPs.

[6] A yellow page is a registry of entries which associate service descriptions to agent IDs [2].

Fig. 5 shows (with the arrow labeled "Move") an active bundle being disseminated from its creator in *Container 1—Client Application* to *Container 2—AB Destination (ABD)*. In ABTTP, the destination is specified when an active bundle is created by AB Creator. Details about the containers follow.

6.3.1. Main Container—AB Coordinator

Main Container—AB Coordinator hosts DF, which provides a yellow-page service. The service is used by mobile agents to register and deregister their services, to modify their registered service descriptions, and to search for needed services.

In this prototype, DF registers four agents: the single active bundle, and three AB services: Security Services Agent (SSA), Trust Evaluation Agent (TEA) and Audit Services Agent (ASA). The agents communicate in such a way that messages exchanged between them are delivered to a destination agent even when it moves around, visiting different hosts. A non registered agent cannot receive messages addressed to it because its location (its host and its container) cannot be found.

6.3.2. Container 1—Client Application

Container 1—Client Application hosts AB Creator. First, AB Creator accepts a user's input for an active bundle—including sensitive data, metadata, and the indication of the destination for the bundle. It creates an active bundle, filling its structure with the contents provided by a user. Second, AB Creator includes a set of functions (which are the code of the active bundle class) that compose the bundle's VM. (The user does not input the code for the active bundle. Instead, since an active bundle object is an instance of the active bundle class, the object uses the code of the class). Next, it registers the complete bundle with DF.

In the prototype we use only a subset of the elements of the structure of an active bundle. This is consistent with the role of the prototype: it is built to prove feasibility of the active bundle scheme.

6.3.3. Container 2—Active Bundle Destination

The function of *Container 2—AB Destination* is receiving active bundles. We can create an arbitrary number of AB Destination instances. Each AB Destination instance must be enclosed in a container that has a unique name (e.g., Container 22, Container 37). Any active bundle can move to any AB Destination (as decided by the AB owner).

6.3.4. Container 3—Active Bundle Services

Container 3—Active Bundle Services includes three service agents: *SSA, TEA,* and *ASA*. The first agent, SSA, maintains a database of identity-related information on active bundles. This information is used for encrypting and decrypting contents of active bundles. SSA keeps the following information on each active bundle: name of the bundle, its decryption key, and the trust level that a host must satisfy to use the active bundle. Note that this information is different from the bundle's metadata. Metadata is contained within the active bundle itself while this security information is stored on the SSA's host. SSA (which is a TTP) stores identity-related data for active bundles in a file as a hash table. We chose the hash table data structure since it speeds accesses to SSA identity-related data.

The second agent, TEA, answers requests from SSA about the current trust level for a specified host. Since our goal is proving feasibility of the active bundle scheme, not developing trust management or trust evaluation solutions (which is a research subject in itself), we do not implement a full-fledged TEA. Instead, we simulate a TEA service by assigning a random number as a response to a request for a host's trust level. In actual active bundle scheme implementations, building TEA will require implementation of trust management based on trust measurement and estimation [31].

The third agent, ASA, monitors activities of active bundles. It receives audit information from active bundles, and records this information into a file for possible analysis by authorized entities[7] (e.g., owners, auditors, or forensic analysis agents). In this prototype, ASA records when an active bundle arrives at a destination host. The audit service in the prototype can be easily extended to facilitate auditing other activities of active bundles, such as access to their data by recipients.

SSA, TEA, and ASA share configuration management data. These data are currently included in a Java class; values for a set of system parameters are specified there, and can be accessed by methods of the class. SSA, TEA, and ASA also share the same *logging* class. The logging tool in ABTTP is a Java class that provides a method to log important events during execution of the prototype for debugging purposes. Log information is output to both a screen window and a log file.

6.3.5. Active Bundles

ABTTP implements an active bundle as a mobile agent with a set of attributes and operations. Note that a mobile agent includes functions that support operations of active bundles. An active bundle is an instance of this mobile agent Java class. Therefore, the bundle includes these functions as the bundle's VM.

6.4 Behavior of the AB Components

Behavior of an active bundle can be summarized as a sequence of three steps:

1. AB Creator *creates* an active bundle. This includes providing the owner with a list of possible destinations for the bundle, and requesting that the owner selects one (details are given in Subsection 6.4.1).
2. Upon receiving the destination choice, the active bundle triggers the *constructing* algorithm. Construction steps for AB includes encrypting and signing data included in it (details are given in Subsection 6.4.2). Then, after receiving its destination from its owner, AB moves[8] to the destination host. Note that an active bundle can construct itself before its destination is given to it.
3. After arriving at the destination host, AB triggers its own *enabling* algorithm (more details are in Subsection 6.4.3).

In the following we describe more details of the algorithms for creating, constructing, and enabling an active bundle.

6.4.1. Creating an Active Bundle

AB Creator creates an active bundle "object" of the Java class "active bundle." This class includes code that becomes the VM of the active bundle when the class is instantiated.

The *creation* process involves an owner of sensitive

[7] As a future extension of this work, we plan to realize a set of tools to support audits.

[8] ABTTP uses the push dissemination style.

data providing AB Creator with sensitive data and metadata. AB Creator creates an active bundle by putting together data, metadata, and adding a VM to the bundle. The bundle now becomes an active entity that can perform its activities using its own VM.

Recall that we assure that a host receiving an active bundle *cannot change* execution of the bundle's VM since it is executed by hosts that have TPM.

6.4.2. Constructing an Active Bundle

The steps of the algorithm for *constructing* active bundles, summarized in Fig. 6, are:

1. An active bundle requests and gets from SSA two pairs of public/private keys. The first pair of keys is used for encrypting data included in it, and the second pair of keys is used for signing the data, or verifying the data signature. The reason for having two key pairs is to prevent attackers (in the case of using only one key) from modifying bundle's sensitive data, and signing it again with the public key of the data owner.
2. The active bundle sends a request to SSA, asking it to store the bundle's *identity-related data*. These data include the name of the bundle, its decryption key, and the minimum required trust level for individual hosts (each host's trust level requirement must be satisfied before it may access the bundle's data). The goal of using SSA is to keep the decryption keys and other identity-related data for active bundles in a trusted location. The decryption keys are issued by SSA only to hosts that are *eligible* to access the bundle, which in this simple prototype implementation means hosts with their trust level exceeding the minimum required by the bundle.
3. The active bundle computes a hash value for sensitive data, and signs them (on behalf of its owner) using signature keys. Signature keys are not disclosed to containers receiving active bundles, and are available only at the client application used by the owner. The signature certifies that data are from its owner.
4. The active bundle encrypts sensitive data using the encryption key.

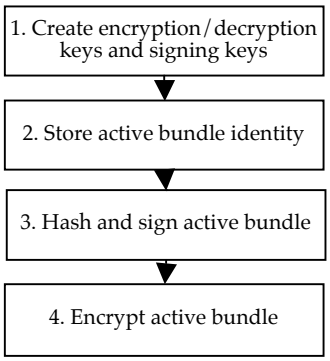


Fig. 6. The UML activity diagram for constructing an active bundle

6.4.3. Enabling an Active Bundle

After arriving at its destination (*Container 2—Active Bundle Destination*), an active bundle *enables* itself. The enabling algorithm for an active bundle, illustrated in Fig. 7 can be summarized as follows.

In Step 1, an active bundle sends a request to SSA asking for its identity-related information.[9] The request includes the identity of the bundle's current container. (The bundle knows the identity of its container because it was specified as its destination when the bundle's move was ordered by the owner.) SSA replies with *all* identity-related information if the destination host has a satisfying trust level. Otherwise, it provides the host's trust level, but not the decryption key.

In Step 2, an active bundle checks if the hosts' trust level is sufficient for accessing the bundle's data. If not, the bundle apoptosizes (Step 3); otherwise, it executes Step 4.

In Step 4, an active bundle checks the integrity of its sensitive data by computing the hash value for these data. Then, it verifies its signed hash value by comparing it to the computed hash value. If the verification fails, the bundle apoptosizes (Step 5); otherwise, the bundle uses the decryption key received in Step 1 to decrypt its sensitive data (Step 6).

In Step 7, the active bundle enforces its privacy policies. In this prototype, we do not develop and use privacy policies. (It is not necessary in this proof-of-feasibility prototype.)

In Step 8, the active bundle provides the output to the visited host. In Step 9, the active bundle sends audit information to ASA. This information includes the bundle's name, its host's identity, and the name of the audited event that is being reported.

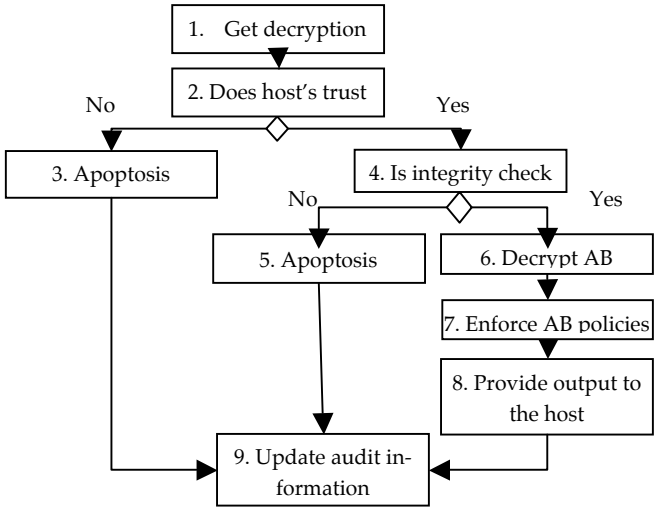


Fig. 7. The UML activity diagram for enabling an active bundle

6.5 Deployment Setup

By deployment setup we mean the distribution of components among hosts and sessions. The login process creates an initial process and a session.[10]

To follow is the deployment setup that we use. We use two computers to run the components of ABTTP. The first computer is a laptop with Windows Vista as the OS. The second computer is a server that runs UNIX as the OS. This deployment setup helps to assure that ABTTP runs on different computers with different operating systems.

Note that with this deployment setup, we regroup SSA, ASA, and TEA in the same container. This choice is

[9] Recall that identity-related information for a bundle includes the decryption key, the signature verification key, and the required trust level.

[10] A session is a collection of processes that were started directly or indirectly by the initial process.

only for convenience since when they are grouped, we activate the three agents by executing the same class. For performance investigation, the three agents could be distributed on hosts and containers differently.

In this setup, we grouped the components into four groups. Fig. 8 shows the deployment diagram of ABTTP. In this deployment, each group of components is located in a different container. The first group, Client

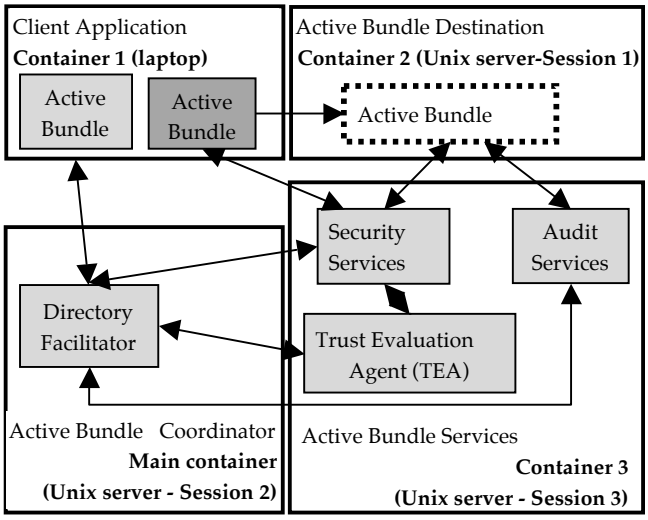


Fig. 8. Diagram of the deployment setup of ABTTP

Application, is composed of AB Creator, and is located in Container 1. It is executed by the laptop. Components of the other containers are executed by the same computer which is the server. The second group, AB Destination, includes the component AB Destination, and is located in Container 2 (Unix Server-Session 1). The third group, AB Coordinator, includes DF (which is a JADE server), and is located in Main Container (Unix Server-Session 2). The fourth group, AB Services, is composed of SSA, TEA, and ASA; it is located in Container 3 (Unix Server-Session 3).

In an experiment with this deployment setup, AB Creator creates active bundles within Container 1. Then the bundles move to Container 2 where they enable themselves.

7 Description of Configuration Setup and Experiments

This section describes how to use ABTTP. Subsection 7.1 describes the configuration of the prototype. Subsection 7.2 describes our experiments with the prototype and Subsection 7.3 discusses the experiments.

7.1. Configuration Setup

The ABTTP prototype is composed of a set of components distributed among hosts, with parameters of the prototype specified by the configuration setup.[11]

The configuration variables are grouped into three groups. Each group of variables is used by one group of components. In the following we describe the three configuration groups used by ABTTP. Table 1 describes the four parameters that the ABC component uses. A value for parameter "LogFile" specifies the name and path of the logging file. A logging file is used by the component to store data that describe the log of activities on an active bundle (e.g., encrypting, decrypting, moving, etc.).

Table 1: Configuration variables for the ABC component

Configuration variable	Description	Example of Values
LogFile	Name and path of the file where the application logs the events.	ABCreatorLog.txt
host	The IP of the host of the Active Bundle Coordinator.	141.001.143.001
Fname	Name of the platform that the client should connect to.	ABFramework
port	The port that the client should connect to in order to communicate with the Active Bundle Coordinator.	1099

Values for parameters "host" and "port" specify the location (IP address), and the communication port of the ABC, respectively. A value for parameter "Fname" specifies the name of the JADE Platform that the component can communicate with. Note that this parameter is not used explicitly by our ABTTP components. It is used by JADE for coordinating the communication between mobile agents.

Table 2 describes the configuration variables for the Active Bundle Services component. The table describes the six parameters that the component uses. A value for parameter "ABIdentitiesStore" specifies the name and path of the file used by SSA to store and retrieve identity-related data about active bundles.

A value for parameter "LogFile" specifies the name and path of the logging file used by the components to store data that describe a log of activities of the services (e.g., receiving messages, sending messages).

A value for the parameter "AuditFileName" specifies the name and path of the audit file. An audit file is used by the component to store data that describe audit information about events on active bundles. In ABTTP we currently record only active bundles' move to ABDs.

Values for parameters "host" and "port" specify the location (IP address), and the communication port of the ABC. A value for the parameter "Fname" specifies the name of the JADE Platform that the components can communicate with.

The difference between a log file content and an audit file content is in the type of data that each file stores. Data in a log file could be used to trace a sequence of events (e.g., for debugging). This data is a temporal data since it is deleted after a time period. The log file could be used by a developer or administrator for debugging.

In contrast, audit file content are data that could be archived and saved for a long time period. The purpose of the audit file is to have a trace of activities that could be used for forensic investigations.

Table 3 describes the configuration variables for the Client Application. The table describes the four parameters variables that the component uses. The variables have the similar description as the one described in Table 1.

[11] The configuration setup determines the values of parameters (a.k.a. configuration variables) used by ABTTP, such as network port numbers, and names of files used by the programs

Table 2: Configuration variables for the active bundles services component

Configuration variable	Description	Example of Values
ABIdentities-Store	Name and path of the file where the information about the identities of created active bundles including keys are stored.	ABkeys.obj
LogFile	Name and path of the file where the application logs events	CoordinatorLog.txt
AuditFile-Name	Name and path of the file where the audit information is stored.	AuditLog.txt
host	The IP of the host of the Active Bundle Coordinator	141.001.143.001
Fname	Name of the platform that the client should connect to	ABFramework
port	The port that the client should connect to in order to communicate with the Active Bundle Coordinator	1099

Table 3: Configuration variables for the active bundle client component

Configuration variable	Description	Example of Values
LogFile	Name and path of the file where the application will log events	ABHostLog.txt
host	The IP of the host of the Active Bundle Coordinator	141.001.143.001
Fname	Name of the platform that the client should connect to	ABFramework
port	The port that the client should connect to in order to communicate with the Active Bundle Coordinator	1099

7.2. Description of the Experiments

This section describes the two experiments that we run on ABTTP. In these experiments, ABC prompts the owner for the name of the active bundle, the sensitive data, the metadata, the required trust level, and the role of the destination hosts.[12] The component initializes and constructs an active bundle and gives it the input of the user. Note that in these experiments the owner selects the destination of the active bundle.

Upon receiving the index of the destination container, the active bundle constructs itself. Then, it moves to the destination container. After arriving into the destination container, the active bundle enables itself. When an active bundle enables itself, it may decrypt itself (Case 1) or apoptosize (Case2). Case 1 is described in subsection 7.2.1 and Case 2 is described in subsection 7.2.2.

To keep the prototype simple, TEA generates trust level values between 0 and 5. These values are distributed based on the normal distribution [1].

7.2.1 Experiment 1: Decryption of an Active Bundle

In this experiment we select value "2" as the required trust level to access the active bundle. TEA generates value "4" as the trust level of the host of the ABD. Therefore, the answer to the question "Does the host's trust level exceed the required minimum?" is "Yes" (see Fig. 7). So, the host is allowed to access some or all sensitive data included in the bundle. Then, the VM of the bundle decrypts its sensitive data (and the visited host can see them).

a) Active bundles services. The value of the required trust level is 2 and the value of the host's trust level is 4.

b) Active Bundle Destination. Note that the active bundle in this case is decrypted.

Fig. 9. Sample screen shots for the executing prototype for Experiment 1

Fig. 9.a displays the log of AB Services, including a request for a hosts's trust level and SSA's response. Fig. 9.b displays the log of ABD, including the records of a successful decryption and an integrity check of the bundle.

7.2.2. Experiment 2: Apoptosis of an Active Bundle

In this experiment, the required trust level to access the active bundle is set to value "2". TEA generates value "0" as the current trust level of the host with the destination container. Therefore, the answer to the question "Does the host's trust level exceed the required minimum?" is "No" (see Fig. 7). So, the host is not allowed to access the bundle's data. Therefore, the threatened bundle apoptsissizes.

Fig. 10.a displays the log of AB Services, including records of a request for a hosts's trust level and the SSA's response. Fig. 10.b displays the log of AB Destination, including the record of bundle's apoptosis.

7.3. Discussion of the Experiments

The active bundle approach provides three levels of confidentiality protection for the bundle's sensitive data throughout their lifecycle. Having three levels improves the protection significantly.

The first level of confidentiality protection relies on using a TTP. TTP has two major roles:

1. Assure that a destination host has a minimum level of trust as required by the owner of an active bundle. TTP supports SSA and TEA.

[12] Note AB Creator requests from the owner the role of the AB D (in the sense of role-based access control) but we do not use this information further

```
SecurityServerAgent-- RegisterABidentity performed on identity of ( agent-id
entifier :name ActiveBundle1@ABFramework  :addresses (sequence http://css01.
cs.wmich.edu:7778/acc http://css01.cs.wmich.edu:38940/acc http://css01.cs.wm
ich.edu:51327/acc http://10.80.148.211:7778/acc ))
SecurityServerBehavior--RegisterABKey
SecurityServerBehavior--securityagent received msgcontent: ActiveBundle1
SecurityServerBehavior--conversationID:RequestDecryptionInformation
SecurityServerBehavior--ab ActiveBundle1
SecurityServerBehavior--send message  My trust?
------------------------
Time: 2010/12/10 21:45:23 Active Bundle: ActiveBundle1 Source:10.80.148.211
Current: 141.218.143.147
TrustServerBehavior--TrustServerBehavior received msgcontent: My trust?
TrustServerBehavior--reply with trust value 0
SecurityServerBehavior-- Received from trust server -- 0
trust0
SecurityServerBehavior--send reply RequestDecryptionInformation-- ((action (
agent-identifier :name securityServer@ABFramework) (ABIdentityItem :ABName A
ctiveBundle1 :ABPublicKey "" :ABRole "my role" :ABRequiredTrust 2 :ABHostTru
st 0)))
```

a) Active bundles services. The value of the required trust level is 2 and the value of the host's trust level is 0

```
ActiveBundleCreateBehavior-- Print of AB after encryption
CipherTools--Print of an active bundle
..-- ABname:ActiveBundle1
..-- Sensitive data:ê!á[påYó«
J*>jð#                          ¬¬
..-- MetaData:0åòùNãinPÔFÃñâ7
..-- Required trust:2
..-- Role:my role
..-- Hostdestination:null
..-- TrustServer:null
..-- SecurityServer:null
..-- SignatureAlgorithm:DSA
..-- SignedHash:[B@4090c06f
CipherTools-- End Of Active Bundle
PuTTYActiveBundleActivateBehavior-----ActiveBundleActivateBehavior --Action
ActiveBundleActivateBehavior--Send Audit information about ActiveBundle1 to
Audit Agent
ActiveBundleActivateBehavior--Send a message to the security server RequestD
ecriptionInformation
ActiveBundleActivateBehavior--reply: ABIdentityItem@366782a9
ActiveBundleActivateBehavior-- Apoptosis due to trust level
ActiveBundleActivateBehavior-----Apoptosis for active bundle ActiveBundle1@A
BFramework on host:141.218.143.147
ActiveBundleActivateBehavior-----Deletion Done ...
ActiveBundleActivateBehavior--Activation done...
ActiveBundleAgent-- Agent shutdown
ActiveBundleAgent--ActiveBundleAgent ActiveBundle1@ABFramework terminating.
```

b) Active Bundle Destination. Note that the active bundle in this case commits apoptosis.

Fig. 10. Sample screen shots for the executing prototype for Experiment 2.

2. Provide decryption keys for accessing bundle's data only when the trust level of the destination host is sufficient to satisfy requirements imposed by the active bundle.

These roles assure following an owner's access control policies for disseminated active bundles throughout the bundles' lifecycle, even when the bundles are far away from the owner (We believe that use of data disclosed by active bundles is based on trustworthiness of host receivinf the data).

The second level of confidentiality protection relies on having different confidentiality protection levels for different portions of data within an active bundle. A decryption key received by an active bundle from TTP can enable a visited host's access to only a portion of bundle's sensitive data; privacy policies decide which portions of data can be accessed.

The third level of confidentiality protection relies on using the bundle's VM. The VM receives needed decryption keys from TTP. Depending on the visited host's trust level (insufficient or sufficient), the VM activates one of the following two actions: (1) apoptosis (for an insufficient host's trust level), or (2) decryption of sensitive data and enforcement of the bundle's privacy policy w.r.t. portions of data (for the sufficient host's trust level).

Recall that the use of VMs combined by TPM assures that the VM is excuted correctly and enforce the privacy policies included in the active bundle.

8 Conclusions

The main challenge in information sharing is the limitations of mechanisms for protecting confidentiality of sensitive data throughout their lifecycle.

We propose in this paper the use of active bundle scheme to protect shared sensitive data. The active bundle scheme is based on attaching to data and metadata a virtual machine (inseparable from data and metadata) which includes enforcement and protection mechanisms.

We describe ABTTP—our TTP-based prototype implementation of the active bundle scheme (where the TTP plays the role of a security server). The TTP assures that a destination host for an active bundle is allowed to access its data only if it has a minimum level of trust as required by the owner of the bundle. ABTTP uses a numerical value indicating the trust level of the visited host.

ABTTP proves that data can protect its confidentiality, which is the contribution of the paper. The prototype demonstrates:(1) encrypting sensitive data and storing decryption keys at a TTP; (2) signing data to ensure their integrity; (3) activating apoptosis when a host receiving the bundle is not allowed to access any portion of bundle's data due to its sufficient trust level; (4) decrypting data, checking integrity of data and simulating enforcement of privacy policies when the receiving host is allowed to access a portion or all data; (5) collecting audit information and storing it by Audit Service Agent (ASA) on a TTP.

Our future work for improving capabilities of ABTTP includes the following:

1. Develop mechanisms for trust evaluation of destination hosts.
2. Extend ABTTP with a formal language to specify privacy policy and mechanisms to enforce the policies.
3. Evaluate the performance of ABTTP in different deployments setups. We will investigate how changing the distribution of ABTTP components among hosts affects the performance of the prototype.
4. Analyze the security of the AB scheme. In this planned work we will evaluate data leaks from active bundles, and study attacks on active bundles.

We are also planning future work for improving the active bundle scheme. We plan to develop an obfuscation approach to obfuscate the code of the active bundle, i.e., a virtual machine. We are investigating an approach for obfuscation based on may-alias. In particular we plan the following: (1) propose obfuscation that use dereferencing, function pointers, and recursive functions; and (2) investigate the efficiency of the construction. This choice is motivated by the fact that alias analysis techniques provide approximation of memory allocation when dereferencing, function pointers, and recursive functions are used.

ACKNOWLEDGEMENTS

This research was supported in part by the NSF grants IIS-0242840 and IIS-0209059, Western Michigan University (WMU) FRACAA award. It is also supported by AFRL under contract number FA8750-10-2-0152.

REFERENCES

[1] J. Banks, J. Carson II, B. Nelson, D. Nicol, "Input Modeling," chapter in *Discrete-event System Simulation,* Fourth edition, Pearson Prentice Hall, Upper Saddle River, NJ, 2005, pp. 307-346.

[2] F. L. Bellifemine, G. Caire, D. Greenwood, *Developing Multi-Agent Systems with JADE*, John Wiley & Sons Ltd, West Sussex, England, 2007.

[3] L. Ben Othmane and L. Lilien, "Protecting Privacy in Sensitive Data Dissemination with Active Bundles," Proc. *7th Annual Conference on Privacy, Security and Trust, 2009 World Congress on Privacy, Trust and Management of e-Business,* Saint John, New Brunswick, Canada, Aug. 2009, pp. 202-213.

[4] L. Ben Othmane, "Active Bundles for Protecting Confidentiality of Sensitive Data throughout Their Lifecycle," Ph.D. Thesis, Western Michigan University, Kalamazoo, MI, Dec. 2010.

[5] W. Bagga, R. Molva, "Policy-Based Cryptography and Applications," Proc. *of the 9th International Conference on Financial Cryptography and Data Security* (*FC 05*), Roseau, The Commonwealth Of Dominica, Feb.2005, pp. 72—87.

[6] D. Braun, J. Sivils, A. Shapiro, J. Versteegh, "Unified Modeling Language (UML) Tutorial," accessed in June 2010. Online at: http://atlas.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/index.htm

[7] M. Casassa Mont, K. Harrison, M. Sadler, "The HP Time Vault Service: Innovating the Way Confidential Information is Disclosed, at the Right Time," Sep. 2002. Online at: http://www.hpl.hp.com/techreports/2002/HPL-2002-243.pdf

[8] H. Chang and M.J. Atallah, "Protecting Software Code By Guards," CERIAS Tech Report 2001-49, Purdue University, West Lafayette, IN, accessed in Dec. 2009. Online at: https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/2001-49.pdf

[9] G. Coker, J. Guttman, P. Loscocco, A. Herzog, J. Millen, B. O'Hanlon, J. Ramsdell, A. Segall, J. Sheehy, and B. Sniffen, "Principles of Remote Attestation," *International Journal of Information Security*, accepted.

[10] Y. Chou, D. Ko, and H. Cheng, "Mobile Agent-based Computational Steering for Distributed Applications," *Concurrency and Computation: Practice and Experience*, vol.21(18), Aug. 2009, pp. 2377-2399.

[11] J. O'Donoghue and J. Herbert, "Data Management System: A Context Aware Architecture for Pervasive Patient Monitoring," Proc. *3rd International Conference on Smart homes and health Telemetric (ICOST*). Sherbrooke, QC, Canada, July 2005, pp.159—166.

[12] "Eclipse.org Home", accessed on June 2010. Online at: http://www.eclipse.org/

[13] S. Montero, P. Díaz, I. Aedo, J. M. Dodero, "Towards Accountable Management of Identity and Privacy: Sticky Policies and Enforceable Tracing Services," Proc. *of the 14th International Workshop on Database and Expert Systems Applications* (DEXA), Prague, Czech Republic, Sep. 2003, pp. 377–382.

[14] "java.con:Java+you," accessed on June 2010. Online at: http://www.java.com/en/

[15] G. Karjoth, M. Schunter, and M. Waidner, "Platform for Enterprise Privacy Practices: Privacy-enabled Management of Customer Data," Proc. *of the 2nd international Conference on Privacy Enhancing Technologies* , San Francisco, CA, Apr. 2002, pp. 69-84.

[16] J. Knudsen, *Java Cryptography,* O'Reilly & Associates, Inc, Boston, MA, 1998.

[17] D.B. Lange and M. Oshima,"Seven Good Reasons for Mobile Agent," *Communication of the ACM*, vol. 42(3), Mar. 1999, pp.88-89.

[18] L. Lilien and B. Bhargava, "A Schema for Privacy-preserving Data Dissemination," *IEEE Trans. on Systems, Man, and Cybernetics,* vol. 36(3), May 2006, pp. 503-506.

[19] National Institute of Standard and Technology (NIST), "Mobile Agents, Oct. 2007. Online at: "http://csrc.nist.gov/groups/SNS/mobile_security/mobile_agents.html

[20] eXtensible Access Control Markup Language *(XACML),* Version 2.0; OASIS Standard, Feb. 2005. Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[21] C. Pfleeger, S. L. Pfleeger, "Database and Data Mining Security," chapter in *Security in Computing, Fourth Edition,* Pearson Prentice Hall, Boston, MA, 2007, pp. 335-336.

[22] B. Parno, "Trust Extension as a Mechanism for Secure Code Execution on Commodity Computers," Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, May 2010.

[23] J. Park, R. Sandhu, J. Schifalacqua, "Security Architectures for Controlled Digital Information Dissemination", Proc. *16th Annual Computer Security Applications*, Dec. 2000, pp.224-233.

[24] A. Rogers, S. D. Ramchurn, N.R. Jennings, M.A. Osborne, and S. J. Roberts, "Information Agents for Pervasive Sensor Networks," Proc. *Sixth Annual IEEE Intl. Conf. on Pervasive Computing and Communications* (*PerCom 2008*), Hong Kong, Mar. 2008, pp. 294-299.

[25] O. Sibert, D. Bernstein, and D. Van Wie "DigiBox: A self-Protecting Container for Information Commerce," Proc. *USENIX Workshop on Electronic Commerce*, New York, Jul. 1995 pp. 15-15.

[26] A. Shamir, "How to Share a Secret," *Communication of the ACM*, vol. 22(11), 1979, pp. 612–613.

[27] R. Shirey, "RFC 2828 - Internet Security Glossary," accessed on Mar. 2008. Online at: http://www.faqs.org/rfcs/rfc2828.html

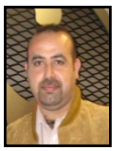
[28] Trusted Computing Group, "Cloud Computing and Security – A Natural Match," April 2010. Online at: http://www.trustedcomputinggroup.org/resources/cloud_computing_and_security__a_natural_match

[29] C. Tschudin, "Apoptosis—The Programmed Death of Distributed Services," Chapter in: J.Vitek and C. Jensen, (Eds.), *Secure Internet Programming*, New York, Spring-Verlag, 1999, pp. 253-260.

[30] P. Vytelingum, T.D. Voice, S.D. Ramchurn, A. Rogers, and N.R. Jennings, "Agent-based Micro-Storage Management for the Smart Grid," Proc. *9th International Conference on Autonomous Agents and Multiagent Systems* (*AAMAS 2010*), Toronto, ON, Canada, May 2010, pp. 39–46.

[31] Y. Zhong, "Formalization of Dynamic Trust and Uncertain Evidence for User Authorization," Ph.D. Thesis, Purdue University, West Lafayette, IN, May 2005.

[32] M.J. Freedman, J. Feigenbaum, T. Sander, A. Shostack, "Privacy Engineering for Digital Rights Management Systems," *Proc. ACM Workshop on Security and Privacy in Digital Rights Management* , Philadelphia, PA, Nov. 2001, pp. 76-105.



Lotfi Ben Othmane received his B.S. in Information Systems from University of Economics and Management of Sfax, Tunisia in 1995; M.S. in Computer Science from University of Sherbrooke, Canada, in 2000; and Ph.D. in Computer Science from Western Michigan University, USA, in 2010. He is currently a postdoc at Eindhoven Technical University, Netherlands. He received many prestigious awards, including the *Best Student Paper Award* from the *Graduate Student Symposium*, *Sixth Annual Conference on Privacy*, *Security and Trust* (*PST 2008*). His research interests include security in data sharing, secure embedded systems, and pervasive computing.

Leszek Lilien received M.E. degree in Electronics/Computer Engineering from Wroclaw University of Technology, Wroclaw, Poland and Ph.D. and M.S. degrees in Computer Science from University of Pittsburgh. He is currently an Assistant Professor of Computer Science at Western Michigan University, Kalamazoo, MI. Dr. Lilien serves on the editorial boards of the *International Journal of Communication Networks and Distributed Systems*; *International Journal of Next Generation Computing*; *Journal of Wireless Mobile Networks*, *Ubiquitous Computing and Applications*; *The Open Cybernetics & Systemics Journal*; and *Recent Patents on Computer Science*. He organized and chaired the *International Workshops on Specialized Ad Hoc Networks and Systems* (*SAHNS 2007*, *SAHNS 2009*, and *SAHNS 2011*), held in conjunction with the *IEEE International Conferences on Distributed Computing Systems*

(*ICDCS*). His research focuses on opportunistic capability utilization networks, a specialized category of ad hoc networks; as well as on trust, privacy and security in pervasive and open computing systems. He has published to date nearly 60 refereed journal and conference papers, and six book chapters.

Bharat Bhargava received the B.E. degree from the Indian Institute of Science and the M.S. and Ph.D. degrees in electrical engineering from Purdue University, West Lafayette, IN. He is currently a Professor of computer science at Purdue University. His research involves mobile wireless networks, secure routing and dealing with malicious hosts, providing security in Service Oriented Architectures (SOA), adapting to attacks, and experimental studies. He is a fellow of the IEEE Computer Society. His name has been included in the Book of Great Teachers at Purdue University. Moreover, he was selected by the student chapter of ACM at Purdue University for the Best Teacher Award.

Mark Linderman received his Ph.D from Cornell University in 1995 and a BSEE from the University of Delaware in 1986. He leads the information management technology area for the Air Force Research Laboratory Information Directorate in Rome, New York. He has worked on reliable distributed repositories, peer to peer systems and data streaming research in recent years. He is working in context/situation aware systems. He is a Senior Member of the IEEE.

Rohit Ranchal received his B. Tech in Information Technology from DAV Institute of Engineering and Technology, Jalandhar, India in 2009. He is a Ph.D student in Computer Science at Purdue University, West Lafayette, IN. His research interests include secure information dissemination, security, privacy, trust in Cloud Computing, and Service Oriented Architecture.

Raed M. Salih received his B.S. in Phusics from Al-Mustanseryah University in 1996; high diploma in computer security from Informatics Institute for Postgraduate Studies in 1999, Baghdad, Iraq; and M.Sc. in computer and data security/information hiding from University of Technology, Baghdad, Iraq in 2002. He worked as a faculty for the Computer Science Department at Al-Mustanseryah University. His research interests include information and data security, privacy policy, and pervasive computing.