CERIAS Tech Report 2015-19
Cross-Domain Data Dissemination And Policy Enforcement
by Rohit Ranchal
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

Graduate School Form 30
Updated 1/15/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Rohit Ranchal

Entitled

CROSS-DOMAIN DATA DISSEMINATION AND POLICY ENFORCEMENT

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

Bharat Bhargava Chair	Leszek T. Lilien
Bradley S. Duerstock	
Samuel S. Wagstaff, Jr.	
Vernon J. Rego	

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Bharat Bhargava

Approved by: Sunil Prabhakar / William J. Gorman 05/26/2015

Head of the Departmental Graduate Program Date

CROSS-DOMAIN DATA DISSEMINATION AND POLICY ENFORCEMENT

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Rohit Ranchal

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2015

Purdue University

West Lafayette, Indiana

To my parents, Mrs. Kamlesh Kumari and Mr. Shakti Ranchal,
and my sister, Dr. Purva Ranchal.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation and profound gratitude to my advisor Professor Bharat Bhargava for his continuous support and guidance. This work would not have been possible without his advice and encouragement. I would like to thank my committee members, Professor Samuel Wagstaff, Professor Vernon Rego, Professor Brad Duerstock, and Professor Leszek Lilien, for their valuable feedback and encouragement. I am grateful to Professor Brad Duerstock for his support during my initial years at Purdue. I am thankful to Dr. Gustavo Rodriguez-Rivera for his inspirational guidance during my teaching assistantships. Under his supervision, I learned a great deal about excellence in teaching. I am thankful to all my current and former labmates, especially Dr. Ruchith Fernando, Dr. Pelin Angin, Dr. Lotfi ben Othmane, Dr. Nwokedi Idika, Norman Ahmed, Denis Ulybyshev, and Zhongjun Jin, for their help and collaboration on various projects. And to Ms. Loveleen Kaur, without whom my decision to pursue doctoral studies would have remained a dream, I am eternally indebted. Last but not least, I am ever so grateful to my family for their unconditional love, tremendous support, and many sacrifices over the years. You all remained a driving force motivating me to accomplish this work with excellence.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Ranchal, Rohit PhD, Purdue University, August 2015. Cross-Domain Data Dissemination and Policy Enforcement. Major Professor: Bharat Bhargava.

Modern information systems are distributed and highly dynamic. They comprise a number of hosts from heterogeneous domains, which collaborate, interact, and share data to handle client requests. Examples include cloud-hosted solutions, service-oriented architectures, electronic healthcare systems, product lifecycle management systems, and so on. A client request translates into multiple internal interactions involving different parties; each party can access and further share the client's data. However, such interactions may share data with unauthorized parties and violate the client's disclosure policies. In this case, the client has no knowledge of or control over interactions beyond its trust domain; therefore, the client has no means of detecting violations. Opaque data sharing in such distributed systems introduces new security challenges not present in the traditional systems. Existing solutions provide point-to-point secure data transmission and ensure security within a single domain, but are insufficient for distributed data dissemination because of the involvement of multiple cross-domain parties.

This dissertation addresses the problem of policy-based distributed data dissemination (PD3) and proposes a data-centric solution for end-to-end secure data disclosure in distributed interactions. The solution ensures that the data are distributed along with the policies that dictate data access and an execution monitor (a policy evaluation and enforcement mechanism) that controls data disclosure and protects data dissemination throughout the interaction lifecycle. It empowers data owners with control of data disclosure decisions outside their trust domains and reduces the risk of unauthorized access.

This dissertation makes the following contributions. First, it presents a formal description of the PD3 problem and identifies the main requirements for a new solution. Second, it introduces EPICS, an extensible framework for enforcing policies in composite web services, and describes its design, implementation, and evaluation. Third, it demonstrates a novel application of the proposed solution to address privacy and identity management in cloud computing.

1. INTRODUCTION

Advances in Information and Communications Technology (ICT) have created a digital economy based on the exchange of information. Information sharing is critical to enhance productivity and maintain competitiveness in the digital economy. Big data is created, processed, shared, and consumed continuously among organizations and people. ICT has been progressing rapidly and with the emergence of models such as cloud computing and advances in mobile technologies, it has become even easier to access and share information. However, the models that facilitate this information sharing make it easier for unauthorized and malicious parties to access private and confidential information. This raises important security and privacy concerns as more and more sensitive data is shared and managed by third-party service providers. Data disclosure without the owner's consent and data breaches have become frequent, aggravating this issue even more. Recent examples include the Target data breach [1], the Anthem data breach [2], and the Sony hack [3], where attackers stole sensitive personal and organizational information. The risks of sensitive data leakage and unauthorized access are among the primary concerns of data owners.

1.1 Motivation

Modern business processes are not self-sufficient, i.e., they focus on their core skills and outsource other activities to specialized service providers, which is more cost effective and improves quality of service. This service-oriented paradigm of distributed collaboration is used in several systems.

- Information Technology (IT) solutions use cloud computing for scalable infrastructure without the upfront investment [4].

- Composite services use external services as components to deliver packaged solutions with broader functionality [5].
- Product Lifecycle Management (PLM) systems and digital supply chains use a complex web of collaboration across globally distributed partners for product development, management, and delivery [6, 7].
- Online cloud marketplaces offer on-demand in-house solutions and third-party API-centric services for rapid application development [8].
- Pervasive healthcare systems use third-party services to store and share medical data across healthcare providers [9].

This service-based model facilitates the integration of independent services from different ownership domains through a loose coupling to achieve a richer system with more sophisticated functionality. It furnishes an easy platform for service providers to dynamically collaborate and form ad hoc applications based on their business requirements, such as to fulfill client requests, meet service level agreements (SLAs), and deliver products and services. To make this complex collaboration work and achieve orchestration goals, information has to be shared across participating entities. Each entity generates, shares, accesses, and uses the data to accomplish its task. As a result, data dissemination is not confined to a single domain controlled by the data owner. Many owners like to track the flow of information to understand how their data is shared and who has access to it. However, the owners of the information have no visibility or control over the interactions in external domains [10, 11]. They lose this knowledge the moment data leaves their domains, as it is difficult to understand and track the access and dissemination of information across multiple entities from different domains.

Figure 1.1 shows an example of data dissemination from a trusted domain to external domain[1] as part of a distributed interaction. Authorized data disclosure and

[1]The external domain is composed of a set of independent domains, each consists of one or more hosts that participate in the interaction. The data owner views the set of domains outside its trust domain as one external domain.

access control become a challenge in such interactions because authentication, authorization, and data dissemination may take place across unknown endpoints that are not visible to the owner. Since the owner is unaware of and has no control over the interactions in external domains, it is not possible for them to track data dissemination. Failure to share data properly can lead to data compromise and unauthorized access resulting in data loss, financial damage, and privacy violation; the owner will have no means of detecting such data leakage [12]. The effect of shared data being compromised is one of the key risks as this data may contain personally identifiable information (PII), medical information, financial information, trade secrets, blueprints, intellectual property, private organizational information, or classified information. Existing approaches are inadequate as they provide point-to-point secure data transmission and mainly protect data inside a single domain, i.e., they do not address the protection of data in a decentralized distributed environment. This thesis proposes a solution that allows secure cross-domain data dissemination based on the evaluation and enforcement of the data owner's policies.

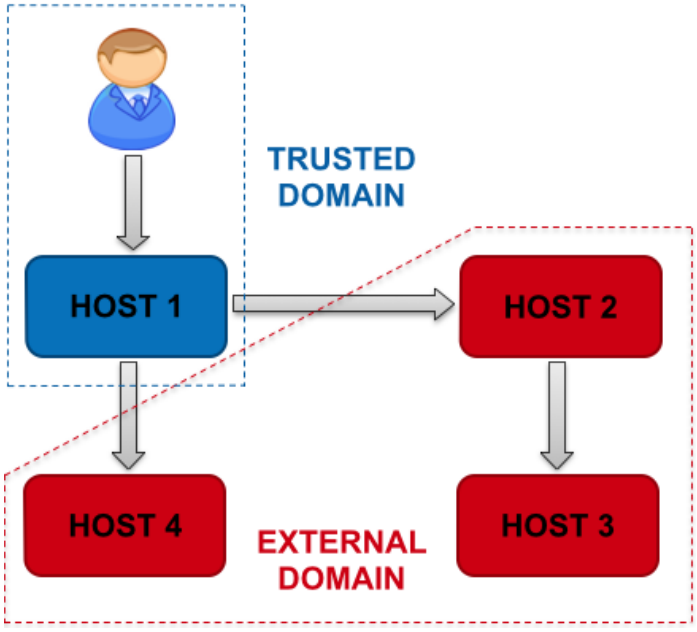

Fig. 1.1. Cross-domain data dissemination.

1.2 Thesis Statement

The thesis statement of this dissertation is:

In a cross-domain distributed interaction, even without the knowledge of the endpoints and the data dissemination path, it is possible to control data disclosure and enforce policies of the data owner at each endpoint.

1.3 Dissertation Contributions

The main contributions of this dissertation are as follows:

- A formal definition and description of the Policy-based Distributed Data Dissemination (PD3) problem and the identification of requirements for a new solution.
- A description of the design, development, and evaluation of a framework for privacy-preserving data dissemination and cross-domain policy enforcement in composite web services.
- A novel application of the proposed solution for privacy and identity management in cloud computing.

1.4 Dissertation Organization

The rest of this dissertation is organized as follows:

Chapter 2. Policy-based Distributed Data Dissemination This chapter provides a formal definition and description of the PD3 problem. It presents an overview of the existing solutions and related work. It identifies and outlines the main requirements for a new solution.

Chapter 3. A Framework for Enforcing Policies in Composite Web Services This chapter provides a description of composite web services and identifies the problems related to a lack of policy infrastructure. It presents the proposed

solution based on the outlined requirements in Chapter 2 and provides details for the realization of the solution. It describes the design and development of a framework for policy transmission, evaluation, and enforcement in composite web services that is compatible with existing service infrastructure. It provides an example scenario to demonstrate the application of the framework and shows the performance and security evaluation under different contexts.

Chapter 4. Identity Management in Cloud Computing This chapter discusses the privacy issues that occur due to the use of cloud-based services and proposes a novel approach to achieve user-centric identity management in cloud computing without using trusted third parties.

Chapter 5. Summary This chapter summarizes the main contributions of the dissertation and presents a discussion of how this work can be extended in the future.

2. POLICY-BASED DISTRIBUTED DATA DISSEMINATION

Distributed systems such as composite web services, cloud-hosted solutions, etc. comprise a number of hosts, which collaborate, interact, and share data. One of the main requirements of these systems is Policy-based Distributed Data Dissemination (PD3) [13]. In the PD3 problem, the data owner needs to share data with a set of hosts. Each host is only authorized to access a subset of data. The data owner can directly interact only with a subset of hosts and relies on them to disseminate that data to other hosts in the system. Note that in order to ensure the correct delivery of appropriate data to each host, it is necessary that each host shares the entire set of data even though the hosts are only authorized for a certain subset of data. We provide a formal description of the problem and propose a data-centric approach to address PD3. The approach enables policy-based controlled data dissemination and protects data throughout their lifecycle. It is independent of trusted third parties, does not require the source availability, and has the ability to operate in unknown environments. This chapter is based on the work described in "Policy-based Distributed Data Dissemination" [13].

2.1 Problem Description

In this section, we describe a data sharing situation, provide a formal description of the PD3 problem, and discuss the goals of the situation.

2.1.1 Preliminary Notions

Definition 2.1.1 *Data disclosure is defined as the process of sharing secure information of a data owner to an entity that requests access to it. It can be authorized based on the policies of the data owner or unauthorized violating the policies.*

Definition 2.1.2 *Data dissemination is defined as a sequence of data disclosures to different endpoints in a distributed interaction.*

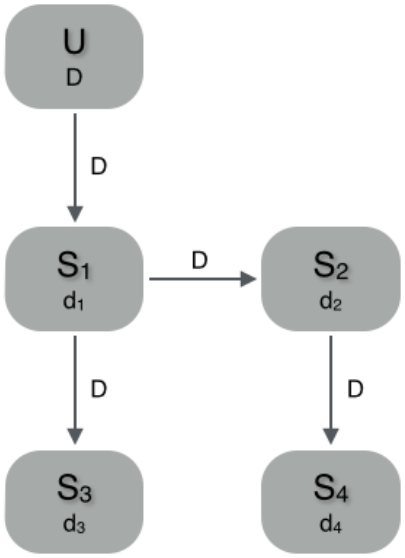

Fig. 2.1. Distributed data dissemination.

Consider a user U, who owns data D that need to be shared with a set of hosts S. Each host S_i is only authorized to access a subset of data d_{S_i}. U is only able to interact with host S_1, as shown in Figure 2.1, and relies on S_1 to disseminate data to other hosts. Similarly S_1 can only interact with hosts S_2 and S_3 and relies on them for further dissemination. Note that in order to ensure the correct delivery of appropriate data to each host, it is necessary that each host shares the entire set of data D even though the hosts are only authorized to access d_1, d_2, d_3, d_4 respectively.

$$D = \{d_1, ..., d_n\} \tag{2.1}$$

$$d_i = < k_i, v_i >,\ [k : key, v : value] \tag{2.2}$$

Let D be a set of data items d_i (as shown in Equation 2.1) owned by U that need to be shared with a set of authorized hosts. Each data item d_i is a key-value tuple of the form $< k_i, v_i >$ (as shown in Equation 2.2). We assume that each host S_i is already aware of the set of item keys k_{S_i} of items d_{S_i}, which it is interested in and could be authorized to access. For instance, an email data item is organized as $< email, abc@xyz.com >$, where *email* is the key element, which is already known to the interested hosts, and *abc@xyz.com* is the value element that should be disclosed only to the authorized hosts. The user U does not have the knowledge of all the hosts and the data dissemination path in advance. It defines access control policies to authorize hosts for data disclosure. Any host that satisfies the conditions defined in the policies for a specific data item is authorized to access the data item.

$$P = \{p_1, ..., p_m\} \tag{2.3}$$

$$AP_i = \{p_a, ..., p_z\}\ or\ \emptyset \leftarrow \tag{2.4}$$

$$P \leftarrow \cup_{i=1}^{n} AP_i \tag{2.5}$$

P is the set of access control policies defined by the user U on data D (as shown in Equation 2.3). An item d_i in data D may have several applicable policies. AP_i is the subset of P, which represents the policies applicable to item d_i (as shown in Equation 2.4).

Note that AP_i can be the $\emptyset$, i.e., there is no applicable policy defined by the user for the particular data item and can therefore be accessed publicly. The union of all sets of applicable policies AP_i is the set P (as shown in Equation 2.5).

$$C = \{c_1, ..., c_n\} \tag{2.6}$$

In order to ensure secure transmission of data and allow fine-grained access, it is necessary for U to encrypt each data item value v_i separately and obtain ciphertext c_i. C represents the set of ciphertexts corresponding to all data item values (as shown in Equation 2.6). The user U and each host S_i share the encrypted data C with the other hosts, as shown in Figure 2.2, instead of sharing the actual data D. Now the problem is that each authorized host must determine how to derive only the data item values from C for which it is authorized.

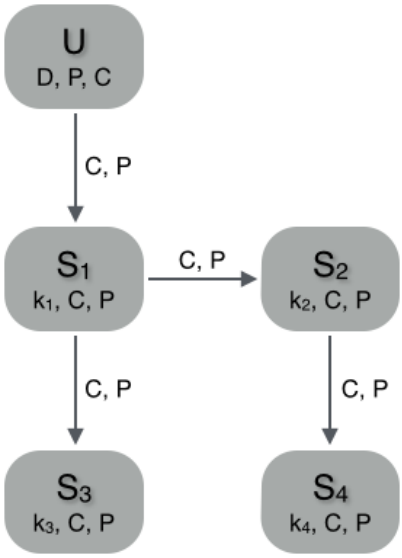

Fig. 2.2. Distributed access control.

2.1.2 Distributed Data Dissemination Goals

The main goals of the data sharing situation are as follows:

1. Data owners should be able to define access control policies for their data items.
2. Hosts should be able to evaluate and enforce the applicable policies to access data items.

3. Each authorized host should only be able to access the data item values for which it is authorized.

4. An unauthorized host should not be able to access any data item value.

2.2 Background

In a trusted domain, it is possible to share the decryption keys along with the policies P and the encrypted data C and trust the hosts to correctly evaluate and enforce the policies, which will ensure the correct disclosure of data to each authorized host. However, in the PD3 situation, the data dissemination path may traverse unknown domains. In these domains, the hosts cannot be trusted to enforce policies for accessing data. In the following, we discuss two categories of existing solutions that can be applied to address PD3.

2.2.1 Cryptographic Solutions

In this section, we discuss three cryptographic solutions that can be used to ensure authorized data access in PD3. These solutions typically rely on a Trusted Third Party (TTP) to issue, manage, and verify the cryptographic keys. The use of TTP introduces issues related to key escrow and poses significant risks of data compromise [14]. TTP must be highly trusted as it could be capable of decrypting data without authorization. It becomes a high-value target and a single point of trust and failure. If the keys of TTP are breached, the data protected by the keys issued and managed by the TTP can also be compromised.

Public-key Encryption

Public-key Encryption (PKE) [15] allows the encryption of data using the recipient's public key such that it can only be decrypted by the recipient using their private key. PKE provides a straightforward solution to address the PD3 problem. Each data

item value v_i can be encrypted using the public key of the host S_i authorized to access this data item value. The encrypted data items C are shared with the hosts and each host S_i can use its matching private key to decrypt only the data item value v_i for which it is authorized. However, there are several issues with this solution. First, the recipient hosts, their public keys, and their authorization levels should be known in advance, which is infeasible for hosts in external unknown domains. Second, the authenticity of the public keys of different hosts must be verified to establish the integrity and ownership of the keys. Third, the data item subsets for which different hosts are authorized should be disjoint otherwise the same data item subset has to be encrypted multiple times with public keys of different hosts, which is not scalable. Fourth, the applicable policies are restricted to a single policy — "if the host has the private key for the requested data item". The use of Public Key Infrastructure (PKI) [16] can avoid some of these limitations. However, PKI uses TTPs to register and verify hosts, and issue and validate keys. In large cross-domain systems, the key distribution and revocation can become highly complex and challenging.

Identity-based Encryption

Identity-based Encryption (IBE) [17] is a type of PKE which allows the encryption of data using an arbitrary string (e.g. unique identity information of the recipient) as a public key. It eliminates the issues of key distribution, authentication, and revocation. However, IBE also requires a TTP for: (a) managing public parameters used by the data owner to derive a public key based on the recipient's identity information; (b) verifying the identity information of the recipient to authorize and issue related private key, which the recipient can use to decrypt the data. The identity information of the hosts should be known in advance and the data item subsets should be encrypted multiple times with the identity information of different authorized hosts. In this case, the applicable policies are limited to the recipient's identity information.

Attribute-based Encryption

Attribute-based Encryption (ABE) [18] is a type of PKE which allows the encryption of data using a set of attributes. The decryption is possible only if the set of attributes of the recipient's secret key match the attributes of the ciphertext. It removes the PKE and IBE limitation of encrypting data items with the keys of all authorized recipients. However, it requires all hosts to be associated with the same universe of attributes and the universe to be known in advance to the data owner, which may not be practical because of the involvement of hosts from different external domains. ABE policies are expressed as boolean and threshold operations over a set of attributes. Such operations have a limited ability to express access control policies and may not be used to design more general authorization systems. ABE also relies on TTP to manage attributes and issue secret keys to the recipient hosts.

2.2.2 Monitoring Solutions

These solutions are based on the execution monitoring of system activities. An Execution Monitor (EM) module runs in parallel with the target system, observes its actions just before their execution, and terminates the system if an action would lead to the violation of policies [19]. The conventional access control mechanisms are modeled based on the monitoring of data access requests to ensure authorized data disclosure. An EM module is setup with the access control policies and installed in the system. The EM is responsible for policy evaluation and enforcement. It intercepts data access requests, evaluates the requests against the applicable policies, and enforces the policies by allowing the access if the policies are satisfied or denying the access if the policies are violated. In a distributed system, the EM is generally installed at, the owner of the data, the recipient of the data, or a mediator, to carry out the policy enforcement. The description of the three cases follows.

Policy enforcement at owner

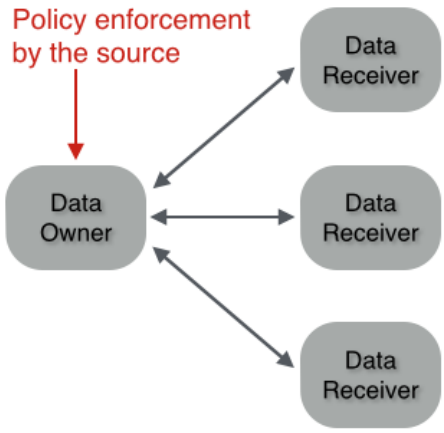

Fig. 2.3. Policy enforcement at owner.

Figure 2.3 shows a typical scenario of policy enforcement at the data owner. In this case, the data owner uses an application that acts as EM to monitor access requests and enforce policies. This is the traditional and most commonly used approach for data sharing in client-server paradigm. Servers employ user authentication before providing a service or sharing data with their users. The main issue with such solutions is that they require the data owner to be visible and accessible to all hosts. The data owner needs to be continuously available to serve access requests. The data owner becomes a single point of failure. A single data owner application may not be scalable as it can become the bottleneck. Given n recipients in the system, the messaging burden for a data owner is $O(n)$ and for recipients is $O(1)$.

Policy enforcement at mediator

Figure 2.4 shows a typical scenario of policy enforcement at a mediator. In this case, a TTP is used as a broker between data owners and data recipients. Data owners send their data and policies to the TTP, which acts as EM and is responsible for enforcing policies and allowing authorized data access requests. For instance, a Pub-

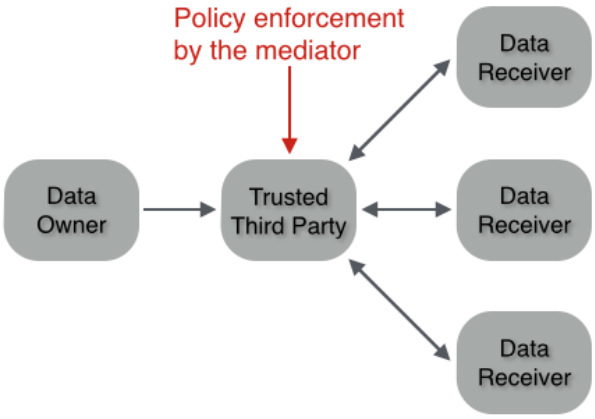

Fig. 2.4. Policy enforcement at mediator.

lish/Subscribe (Pub/Sub) model is based on this approach. In the Pub/Sub model, the subscribers subscribe to the topics of interest at the TTP and are authorized to receive the associated information after the publisher publishes it at the TTP. The use of TTP creates various issues. TTP needs to be trusted for data storage, authorization, and distribution. As a result, there is a loss of control over the data published at TTP. TTP becomes a single point of trust and failure. Data can be compromised in case of external attacks or insider abuse. TTP can aggregate private information and disclose it to interested parties for profit or under subpoenas, court orders, or search warrants. Given n recipients in the system, the messaging burden for a data owner is $O(1)$, for recipients is $O(1)$, and for TTP is $O(n)$.

Policy enforcement at recipient

Figure 2.5 shows a typical scenario of policy enforcement at the recipient. In such solutions, there is a trusted component (hardware or software) present on the recipient host that acts as EM and is responsible for policy enforcement. The access policies associated with the data are either predefined and known in advance or defined during data dissemination. The policies are enforced by the trusted component at the host after the data is received. For instance, digital rights management solutions

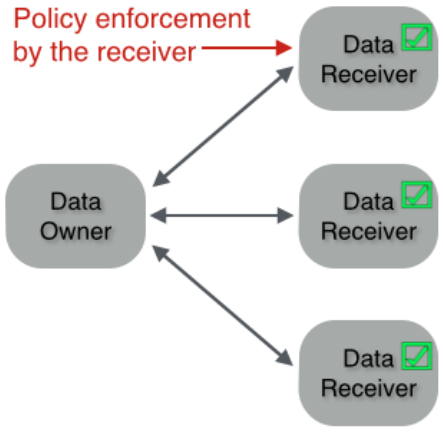

Fig. 2.5. Policy enforcement at recipient.

use this approach to authenticate users and control access to the protected data. This approach requires advance knowledge of the recipient hosts and the presence of a trusted component on the hosts. Dynamic policy communication to external domains is problematic because it requires secure communication and placement of policies in the trusted component. The policies are limited because they need to be pre-defined during the setup of the trusted component. The distribution and setup of the trusted component at the recipient hosts are a challenge. Therefore, this approach may not be used for data dissemination to external domains. Given n recipients in the system, the messaging burden for a data owner is $O(n)$ and for recipients is $O(1)$.

Although trusting a recipient or TTP to enforce policies allows for a relatively straightforward solution, however, there is a large downside to this approach – these entities must faithfully evaluate and enforce the policies, which is an impractical assumption for systems that include external domains. Furthermore, these entities must remain uncompromised to ensure correct policy enforcement, which is becoming increasingly difficult with the attacks and intrusions of services becoming commonplace (cf. recent data breaches at Target [1], Anthem [2], Sony [3]).

2.3 Proposed Solution

In this section, we state the assumptions and propose a new solution that can be used to achieve policy-based fine-grained data dissemination in a distributed environment.

2.3.1 Assumptions

The assumptions regarding the setup are described as follows:

- Item keys k_i are public information and already known to the hosts.
- Each host forwards the required information correctly to the next hosts.
- The information is securely transmitted and not modified during transfer.
- Each host performs the requested operations on the information that it receives.

2.3.2 Solution Overview

We propose a mobile (portable) Execution Monitor (EM) which is transmitted along with the data and policies to each endpoint in the interaction. It authorizes data access requests based on policy evaluation and enforcement and ensures controlled data disclosure at each endpoint. The data owner constructs an Execution Monitor EM and shares it with each host S_i (with which the owner can interact) along with the encrypted data C and the policies P, as shown in Figure 2.6. Using ciphertext data C, policies P, and an item key k_i as input to EM, a host can derive the corresponding item value v_i only if it is authorized to access the data item d_i based on the access control policies AP_i. When a host S_i receives EM, P, and C, it forwards them to other hosts with which it can interact. Each host S_i executes the EM and requests access to the data item value using the item key k_i. The EM evaluates the access request for a data item key k_i against applicable policies AP_i and allows access to the corresponding data item value v_i only if the policies are satisfied. In this case, the

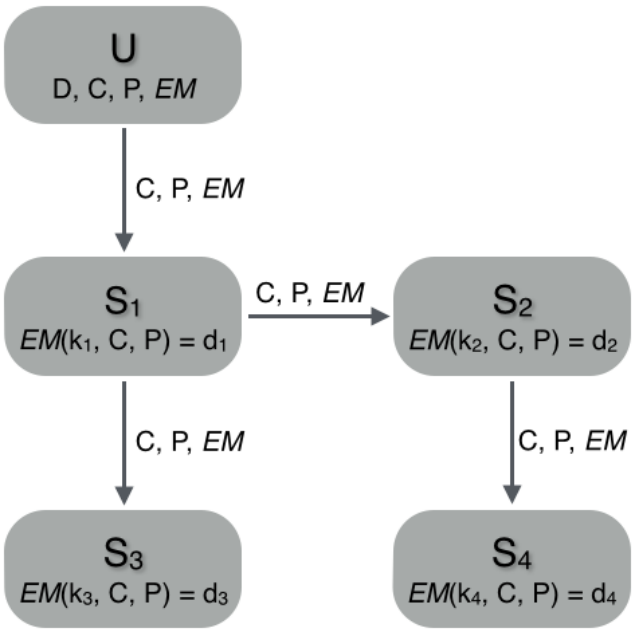

Fig. 2.6. Policy-based distributed data dissemination.

messaging burden is shared across all entities based on the composition topology of the PD3 setup. It can be $O(1)$ for all entities in a chain topology and $O(n)$ for the root of a facade topology.

2.3.3 Solution Requirements

In the following, we describe the main requirements for the correctness of the solution. The EM needs to correctly identify a host, validate its data access request, evaluate and enforce applicable policies, and disclose appropriate data. The main components required for the correct operation of the EM are outlined as follows:

Host Authentication

This component is responsible for verifying the identity of the host. This is achieved using standard authentication mechanisms. These mechanisms are based on one of the following criteria [20]: (a) host knowledge, e.g. passwords; (b) host

ownership, e.g. digital certificates; (c) host inherence, e.g. biometrics. A suitable authentication mechanism can be selected based on the application. The authentication mechanism should be able to correctly authenticate the host and identify a cheating host that pretends to be an authenticated host.

Data Access Authorization

This component is responsible for validating the data access requests by a host based on the evaluation of applicable policies for the requested data items. This is achieved using standard authorization mechanisms such as access control based on: (a) the role of the host (Role-based Access Control (RBAC) [21]), or (b) the attributes of the host, request, or environment (Attribute-based Access Control (ABAC) [22]). The authorization mechanism should be able to correctly evaluate the applicable policies for a data access request and identify the authorization level of the host.

Data Disclosure

This component is responsible for disclosing the appropriate data to an authorized host. Data disclosure includes decryption of data and transmission to the host. The cryptographic keys need to be obtained for data decryption. Note that the keys cannot be obtained from the data owner in cases when the recipient host is not able to directly interact with the data owner. In the following, we discuss four possible methods for key management.

Key inclusion approach sends the keys along with the data. This approach can be suitable for a trusted domain. However, it is prone to attacks and is not feasible for interactions with external domains.

Centralized key management approach uses a TTP to store the keys and distribute them to authorized hosts. For instance, PKI and IBE systems use this approach.

Distributed key management approach avoids the use of a dedicated TTP. An example approach used in the Vanish system [23] works as follows. First, it sets a

threshold and splits the keys using threshold secret sharing technique [24] into multiple shares such that at least a threshold number of shares are required to reconstruct the key. Second, it uses a public Distributed Hash Table (DHT) [25, 26] to store the key shares in different nodes. Third, it reconstructs the key for data decryption by retrieving the threshold or more number of shares. This approach provides better security for key management, but it is unsuitable for real-time interactions involving multiple data decryptions due to the overhead of retrieving the shares and reconstructing the key for each data decryption. The use of a public DHT for storing key shares may not be practical because the shares may disappear due to the node churn occurring as a result of nodes being dynamically joining and leaving the network.

Dynamic key derivation approach is proposed as an alternative. It derives the keys based on the unique information generated during the execution control flow steps of the EM. The information needs to be accurate in order to generate the correct key. The accurate information is only generated if the host is authenticated and the data access request is authorized. A similar approach has been used for data encryption [27].

Tamper Resistance

This component is responsible for the correct execution of the EM and its components. It protects data leakage due to malicious modifications of the EM. It ensures that the correct disclosure of data depends on the correct execution of the control flow steps of the EM. The correctness is checked by verifying the integrity of the EM to ensure that there is no difference from the original code of the EM. Common software tamper resistance approaches are based on guards [28] that utilize secure one-way hash functions to calculate the digests of the software code and its resources [27].

3. A FRAMEWORK FOR ENFORCING POLICIES IN COMPOSITE WEB SERVICES

IT solutions are moving from a traditional monolithic design to a Service Oriented Architecture (SOA) that comprises a number of loosely-coupled independent services to handle client requests [4]. With the emergence of cloud marketplaces, the API-centric service model is rapidly becoming the de facto model for delivering IT solutions and services [5]. This model extends the existing SOA with the ability to dynamically orchestrate composite Web services to cope with business requirements and service level agreements (SLAs). It allows dynamic selection of component services from external domains and enables active service composition reconfiguration to ensure high-quality service delivery. A client request to a composite service translates into multiple secondary service interactions involving component services. A composite service orchestration may invoke unauthorized services and share data, violating client's policies. The client has no means of identifying if a violation occurred because it has no control over the chain of service invocations. In this chapter, we present a framework for Enforcing Policies in Composite Web Services—The EPICS framework. This framework ensures privacy-preserving access control in SOA. This chapter is based on the work described in "A Framework for Enforcing Policies in Composite Web Services" [29].

3.1 Introduction

Modern information systems (such as Netflix [30] and Amazon [31]) are modeled based on service computing because of its many advantages. System components are built as independent services that communicate to handle incoming requests. The resources in these systems are made available through the services. These systems

can handle a large number of interactions and process large amounts of data. SOA is leveraged to increase availability, scalability, and resiliency of the system [4, 32]. This paradigm is used in many different types of information systems such as enterprise business-to-business systems [11], cloud computing systems [30], product lifecycle management systems [6], pervasive healthcare systems [33], digital supply chain systems [7], etc.

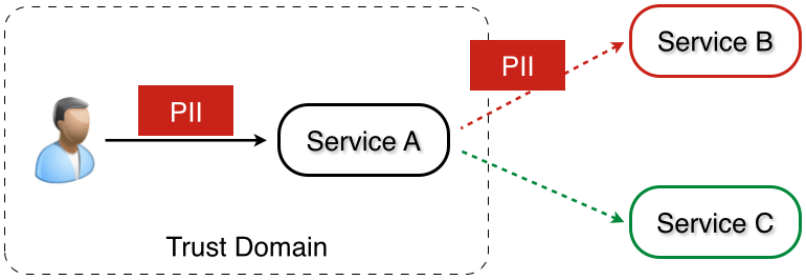

Fig. 3.1. Data leakage in SOA interaction.

Access control remains a challenge in SOA because authentication, authorization, and data dissemination may take place across unknown (possibly untrusted) endpoints. Figure 3.1 shows an example of a SOA interaction involving cross-domain services. In this case, the client's personally identifiable information (PII) is disseminated out of the client's trust boundary, behind-the-scenes. The client interacts with *Service A* after establishing a trust boundary through mutual authentication, authorization, and pre-existing relationships (e.g. a registered user of the service). *Service A* invokes two components: *Service B* and *Service C* to handle the client's request. Note that the next client request (of the same type) may be sent to different component services. The client, the owner of the information, has no knowledge of this service composition, nor any control over the disclosure and dissemination of information that they share with the primary service. The secondary interactions may result in data being shared with multiple unauthorized (from the client's perspective) services and the client has no means of detecting such data leakage. Such dynamic

cross-domain composite service interactions introduce new security challenges which are not present in the traditional SOA systems.

3.1.1 Motivation

A SOA client interacts only with the front-end service (the primary composite service) and has no awareness of the service orchestration that fulfills their request. Thus, the client is unable to predetermine all the services that will participate in the orchestration and have access to their data. Traditional security solutions designed to prevent unauthorized access to resources provide protection in a well-defined perimeter and are not applicable to dynamic cross-domain service interactions. Existing security solutions such as host-level security (e.g. TLS) or message-level security (e.g. WS-Security, HTTPS) provide point-to-point security and are inappropriate for protecting shared data in complex SOA service invocations. These types of solutions provide a secure communication channel to transfer data between two services, but SOA interactions may involve many services unknown to the data owner. Furthermore, certain assumptions, which may be valid in the case of point-to-point interactions, are not valid for composite interactions. The policy compliance assumption is not valid; examples include when service providers use third-party services to fulfill client requests, discarding the required policies or when service providers trust third-party services to enforce the required policies. In a static service composition, the interaction policies can be pre-determined, negotiated, communicated, and placed in the system. Reputation, authentication, and SLAs can be used to establish mutual trust relationships and address accountability. Because composite service orchestrations are dynamically formed depending on the current context[1], the assumptions about policy enforcement in dynamically orchestrated service compositions are not practical.

[1]The context may include the source of request, type of request, category of services, availability of services, charges of services, quality of services, etc.

The clients could have specific requirements (policies) while interacting with different services for their information's disclosure and dissemination. Furthermore, the clients may have specific access policies for individual data items, for e.g., healthcare data can have a patient's medical record intended "only for attending physician at the hospital", a patient's insurance number intended "only for employees of the insurance company", and a patient's prescription intended "only for a pharmacist". Such fine-grained policies require dynamic policy specification, communication, evaluation, and enforcement at each service endpoint. However, the current SOA infrastructure does not support these requirements. These limitations either restrict SOA interactions or cause security and privacy risks to clients. The current SOA model has the following issues:

- **Lack of transparency:** Clients have no visibility in service interactions beyond the primary service. Therefore, they have no means of ensuring that an authenticated entity is accessing only the information for which it is authorized.
- **Loss of control:** Clients are unable to specify and communicate policies for their information, e.g. access control or dissemination control policies. Therefore, they have no means of controlling with whom their information is shared and how it is accessed.
- **Lack of trust:** Widespread service data breach events combined with the lack of visibility and the loss of control lead to diminished trust in the current SOA model.
- **Lack of policy infrastructure:** A trusted infrastructure through which policies can be dynamically communicated and readily evaluated and enforced is required to establish a web of trust in SOA.
- **Threats to client's information:** Sensitive information related to a person such as PII, healthcare records, personal preferences, shopping patterns, lifestyle information, etc. is stored by service providers to increase efficiency,

utility, and provide value added features. Information aggregation can result in user profiling. This information is prone to insider abuse, external attacks, subpoenas, etc. Data leakage to interested parties such as advertisers, attackers, government agencies can result in targeted advertisements, financial losses, and privacy violations.

In order to address the above-mentioned issues, a new mechanism is required that allows the clients to dynamically specify policies, has the ability to communicate policies to each endpoint in the interaction, and ensures dissemination of data based on the enforcement of policies in both the trusted and external domains. In this chapter, we present a framework for policy-based controlled data dissemination in composite Web services. This framework is modeled based on the solution requirements identified in Chapter 2. "Policy-based Distributed Data Dissemination". It uses a novel data-centric approach to transform passive data into an active entity that is able to protect itself. It enables dynamic data disclosure decisions and protects data throughout their lifecycle. The granularity of the data being shared with a service is determined by the evaluation and enforcement of the client's access control policies. An online-shopping scenario is used as an example to demonstrate the working of the framework.

3.2 Background

This section provides a background about composite Web services, access control, active bundles, and software tamper resistance.

3.2.1 Composite Web Services

Composite Web services are an extension of the traditional SOA model, which has evolved with the proliferation of cloud-hosted solutions, mobile apps, and API-centric services. Composite Web services integrate disparate, distributed, and self-sufficient

services through a loose coupling to achieve a larger system with more sophisticated functionality [34]. This architecture enables the dynamic composition of service orchestrations: self-contained, loosely-coupled, and dynamically composed applications to address the business requirements. The services can be independently developed and transparently deployed while maintaining a stable Application Program Interface (API) for their consumers [4]. This allows concurrent use of services from different ownership domains without changes to existing services. Services are reusable and can simultaneously participate in multiple orchestrations. Figure 3.2 shows an example of a service orchestration in a composite Web service for online-shopping. Services are dynamically composed based on the requirements of the client's request, for e.g., the seller service is selected based on the client's order, the payment service is selected based on the client's payment, and the shipping service is selected based on the client's mailing address.

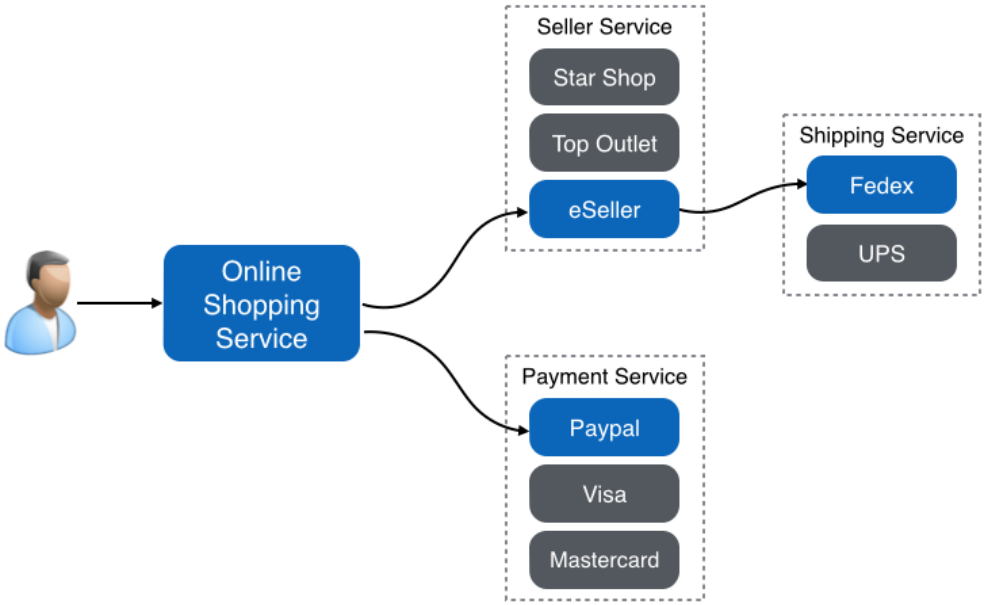

Fig. 3.2. Service orchestration in a composite Web service.

Composite services abstract their internal working complexity and provide a simple interface independent of the implementation of the individual services in their compositions. Services and consumers communicate by passing data in well-defined

message formats over standard Web protocols, such as Simple Object Access Protocol (SOAP) and Representational State Transfer (REST). A service can act both as a service provider or a service consumer depending upon its role in the service orchestration. A service consumer sends a request to the service provider and the provider performs a unit of work on behalf of the consumer by utilizing the data shared by the consumer in the request.

3.2.2 Access Control

Access control is a security aspect that is used to restrict unauthorized access to shared resources based on well-defined policies. An access control solution performs three operations [35]: (a) *authentication* is used to verify the identity of the entity requesting access to resources, (b) *authorization* is used to validate the permissions of an entity to access a particular resource by evaluating the applicable policies against the access request, (c) *accountability* is used to record access control decisions and keep track of access to resources. Several access control models have been proposed such as Mandatory Access Control (MAC), Discretionary Access Control (DAC), Role-based Access Control (RBAC), Attribute-based Access Control (ABAC), etc. Here we discuss the two commonly used access control models.

Role-Based Access Control (RBAC)

In RBAC, the entities requesting access to resources have defined roles [21]. These roles are associated with permissions to access resources. Under this model, an entity is granted access to a resource only if its role has the permissions required to access the resource.

Attribute-Based Access Control (ABAC)

In ABAC, the entities are granted access based on the evaluation of policies [22]. These policies define rules by combining the attributes of subjects (entities requesting access to resources), resources (objects to which access is requested), actions (type of access, e.g. read or write), and environment (context of the request, e.g. time of the access request). Under this model, an entity is granted access only if it satisfies the attributes defined in the policies for the requested resource. The use of attributes allows to express more complex access control policies and enables fine-grained policy-based authorization.

Policy Evaluation and Enforcement

Access control rules are specified as policies. In order to ensure proper access control, these policies need to be evaluated and enforced. The conventional policy evaluation and enforcement mechanisms are based on the monitoring of access requests using an Execution Monitor (EM) module [19]. The EM intercepts access requests, evaluates the applicable policies, and enforces the policies by allowing access if the request satisfies the policies or denying the access if the request leads to the violation of policies. Basin et al. extended the formal model of the execution monitoring by proposing to distinguish controllable actions, i.e., the actions that can be stopped from execution, and observable actions, i.e., the actions that can only be observed, but cannot be prevented from execution by the EM [36]. In this model, the access control actions fall in the category of controllable actions and, therefore, can be controlled by enforcing the policies. Policy enforcement is generally done at the owner of the resource, at the recipient of the resource, or at a mediator. A detailed discussion of policy enforcement models is presented in Section 2.2.2 in Chapter 2.

eXtensible Access Control Markup Language (XACML)

XACML defines a standard specification to express access control policies in XML and a model to evaluate access requests against the policy rules [37]. It can be used to implement both RBAC and ABAC. The main components used to support access control using XACML are as follows [54]:

- Policy Administration Point (PAP) is used to manage access control policies.
- Policy Decision Point (PDP) is used to evaluate access requests for a resource against applicable policies.
- Policy Enforcement Point (PEP) is used to intercept access requests for a resource, send requests to PDP for evaluation, and approve or deny them based on the evaluation results of the PDP.
- Policy Information Point (PIP) is used as a repository to store the attribute values of the entities in the system.
- Policy Retrieval Point (PRP) is used as a repository to store the access control policies of the system.

3.2.3 Active Bundle

Traditional solutions for protecting access to data consider data as passive entities that are unable to protect themselves. They require another active and trusted entity to protect them–a trusted processor, a trusted memory module, a trusted application, or a trusted third party. We challenge this assumption and propose a solution based on the Active Bundle (AB) construct that transforms passive data into an active entity. An AB is a self-protecting data encapsulation mechanism composed of sensitive data, policies, and an engine for policy enforcement [38,39]. The description of the structure of an AB, its types, states, and features follow.

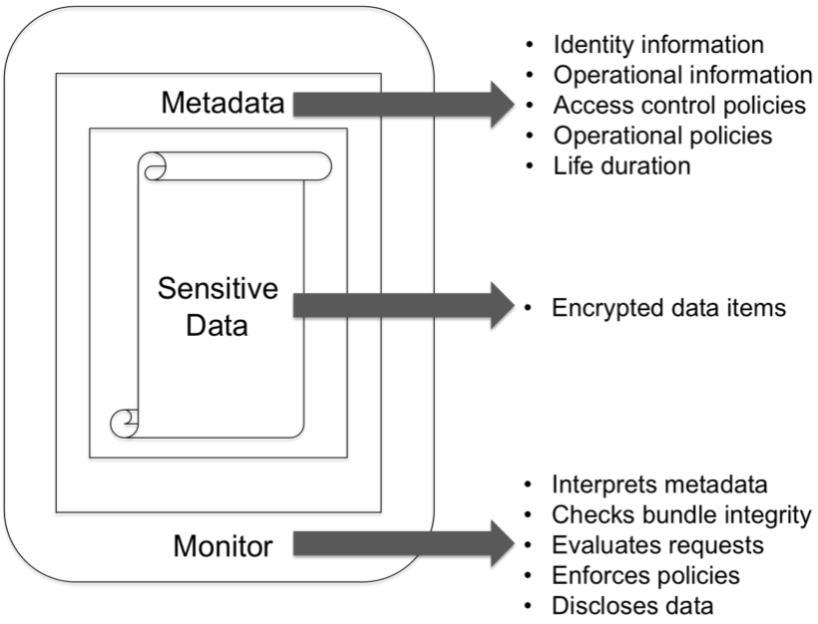

Fig. 3.3. Structure of active bundle.

Structure of Active Bundle

Figure 3.3 shows the structure of an AB. The main components of an AB are as follows:

- *Sensitive Data* is the information that needs to be protected. It can include any digital content such as documents, code, images, audio, video etc. This data content can have several items, each with different protection requirements and applicable policies to ascertain its distribution and usage. For instance, ABs shared among UAVs can contain information with different security classification levels, for e.g., mission critical information with "top secret" classification level and terrain information with "confidential" classification level [40].

- *Metadata* includes a variety of policies that control the behavior of AB and govern its data dissemination. For instance, operational policies that control AB's operation such as expiration time, active time, maximum requests, life

duration, etc. and access control policies that authorize data access requests and address data privacy.

- *Monitor* is a specific purpose program that is used to operate AB, protect its content, and enforce policies to guarantee the appropriate access control to sensitive data of the bundle (e.g. disclosing to a service only the portion of sensitive data that it requires to provide the service).

In this dissertation, we extend the existing AB construct by defining its actions, types, and states as follows.

Actions of Active Bundle

- *Authentication*: It is performed to verify the identity of the service requesting access to data. It can be based on standard authentication mechanisms, such as passwords, certificates, biometrics, etc.

- *Authorization*: It is performed to validate the data access request of the service. It allows or denies access to the data based on the evaluation of the applicable policies. It can be based on the standard authorization mechanisms such as RBAC and ABAC.

- *Integrity Verification*: It is performed to dynamically verify the correctness of the AB's execution. If any modification is detected in the AB's data, policies, or code, the data access request is denied.

- *Data Disclosure*: It is performed to decrypt the requested data and disclose it to the service only if the service is authenticated and its request is authorized. It identifies the encrypted data item and determines the appropriate key to decrypt it.

- *State Exchange*: It is performed to store state information such as interaction logs and retrieve information related to the environment such as the trust value

of a service or current system context. A TTP can be used to exchange this information.

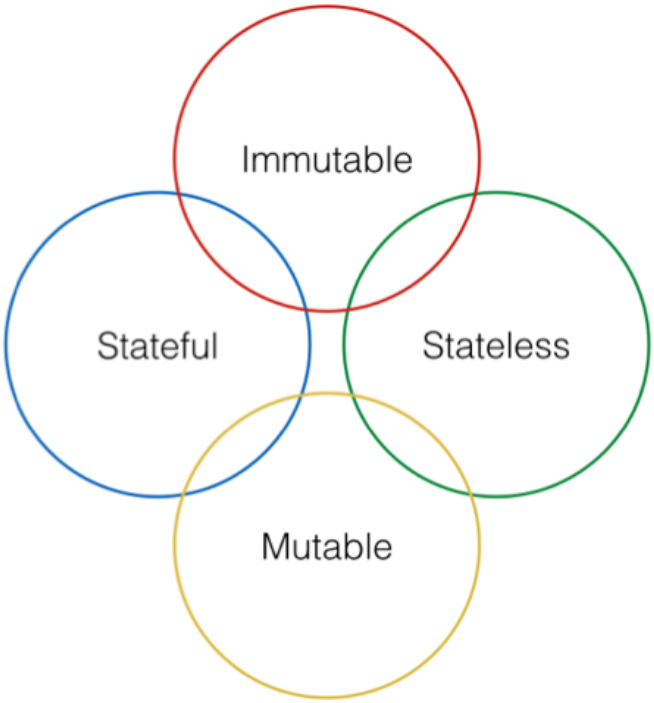

Fig. 3.4. Types of active bundle.

Types of Active Bundle

Figure 3.4 shows a Venn diagram for the types of AB. An AB could be either mutable or immutable, and either stateful or stateless. The four types are described as follows:

- *Immutable AB*: It consists of static data and policies that cannot be modified after creation. Such a structure offers better security and prevents inconsistencies among duplicate copies. It can be either stateful or stateless. An immutable-stateful AB can use a third party service for storing and retrieving state information, interaction data, and logs.

- *Mutable AB*: It can be modified and its data and policies can be updated after creation. The update operation can be performed by AB itself or by a third party. It can be either stateful or stateless. A mutable-stateful AB can locally store state information, interaction data, and logs and carry them to the next endpoint. AB can use this information during subsequent executions.

- *Stateless AB*: It executes in a self-sufficient way and does not maintain any state information about its interactions. It does not depend on a third party service for any information and uses its inherent knowledge to make data disclosure decisions. It considers every interaction as independent and does not keep track of auditing and logging information about its interactions and the information flow path. It can be either mutable or immutable.

- *Stateful AB*: It communicates with a third party service to exchange state information and store its interaction logs. It can request information from the third party service and use it to for policy evaluation and data disclosure decisions. It can be either mutable or immutable. In this structure, the AB has more capabilities such as data provenance and context-aware data disclosure.

States of Active Bundle

The AB states can be classified as either active or passive. Figure 3.5 shows the states of AB. These states are described as follows:

- AB in *create*: This is a passive state. In this state, AB is being created by specifying the data, the policies, and the monitor program to evaluate and enforce the policies and perform data disclosure decisions. From this state, AB jumps into the rest state.

- AB at *rest*: This is a passive state. In this state, AB exists in the form of a file on the filesystem of the host. From this state, AB can either jump into the transfer state or the listen state.

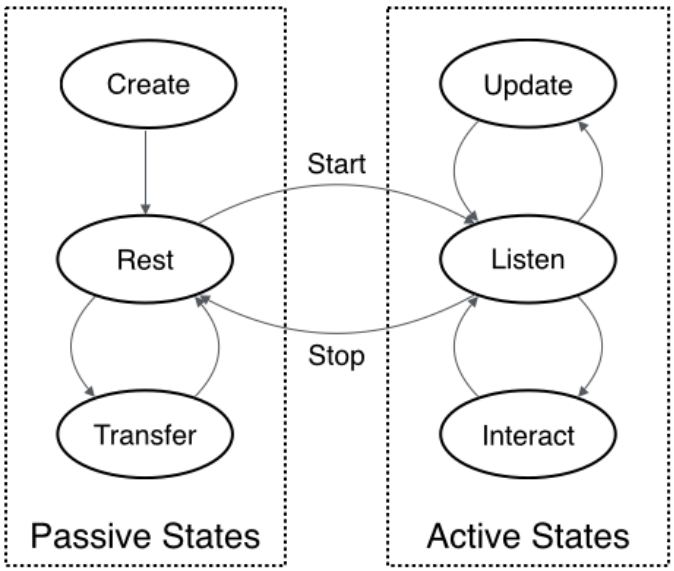

Fig. 3.5. States of active bundle.

- AB in *transfer*: This is a passive state. In this state, AB is encapsulated in a specific message format depending upon the SOA environment and transferred from one host to another host. From this state, AB jumps into the rest state.

- AB in *listen*: This is an active state. In this state, AB is running and listening for incoming data access requests. From this state, AB can either jump into the interact state or the rest state.

- AB in *interact*: This is an active state. In this state, AB is interacting with a service and evaluating the data access request. It allows access to data only if the request is authorized. The evaluation is based on service authentication and access request authorization using applicable policies. From this state, AB jumps into the listen state.

- AB in *update*: This is an active state. In this state, AB's data and policies can be updated and state information such as interaction data and logs can be stored in AB or exchanged with a third party service. From this state, AB jumps into the listen state.

Features of Active Bundle

The main features of AB are as follows:

- Policy-based access control: AB metadata includes policies that enable fine-grained access control to sensitive data.
- Privacy-preserving selective data dissemination: AB has the ability to control data disclosure and provide selective access to data based on different factors such as the host's authorization level, the policies associated with the host, the policies associated with the data requested, etc. In contrast to other approaches, AB tends to minimize data disclosure.
- Context-aware adaptable data dissemination: AB can request environment information (such as system context or trust level of a service) from a third party and update its policies to make context-aware decisions. For instance, in case of a special context like emergency, it can loosen its policies and allow data disclosure; in case of a low trust level of service, it can tighten its policies and deny data disclosure.
- Independent of third party data and policy management: The sensitive data and policies are always held by the AB or the authorized entities and are never disclosed to any third party. Data security is provided through AB, which encapsulates the data instead of using other entities to handle and protect the data.
- Independent of source availability: The data owner does not need to be available after the initial transfer of AB. Once it creates the AB and sends it to a service, the service can transfer AB to other services.
- Ability to operate in the external domains: AB includes a monitor module that intercepts data access requests. Therefore, it can authenticate and authorize interactions in external domains and ensure correct data disclosure. For instance,

a simple blacklist-based policy can be used to control interactions and prevent unauthorized access to data.

- Reduced service liability for protecting private information: AB minimizes data disclosure and only allows authorized access to data. The use of data owner's policies for disclosure decisions reduces the liability of a service for protecting private data and the related legal consequences of data leakage.

- Improved interaction visibility: AB can exchange information about different events in its lifecycle. Besides auditing, this is useful for reporting a malicious activity. This information can be used for detecting attacks on services and evaluating service trust levels. Thus, it improves transparency and accountability.

- Quantifiable Data Dissemination: With this capability, the AB is able to associate a measure with the amount of data disclosed and decide to further disclose or deny more data requests based on its policies. This allows AB to track the degree of privacy with respect to each recipient.

3.2.4 Tamper Resistance

Tamper resistance ensures that a program executes as intended, even in the presence of an adversary who tries to monitor, disrupt, or modify the execution. When a tamper is detected, the tamper resistance mechanism either stops the execution and optionally reports it, or dynamically corrects the modification and continues the execution. Common tamper detection methods include [41]: (a) *code introspection* based on the hash value comparison of the original code with the current version of the code; (b) *output verification* based on the comparison of a computation's result with a preset value or range; (c) *environment validation* based on the monitoring of the execution platform to verify its trustworthiness. The *output verification* method is not sufficient when the tamper does not affect the expected output. The *environ-*

ment validation method is difficult to implement in external environments where the monitoring has no control over the execution platform, and the inspection cannot be fully trusted. Due to these restrictions, *code introspection* is the most suitable method for tamper resistance in a cross-domain service environment.

Code introspection methods augment the program with code that computes the hash over different regions of the program and compares it to expected values [41]. These methods are prone to pattern-matching attacks; the attacks identify the code locations in the program that check the hash and modify the response actions or replace the hash with a pre-computed value to continue execution. Such attacks can be prevented by verifying the integrity of the tamper-resistance code itself. The method of *software guards* suggests that it is not sufficient to check the integrity of the program regions by using the introspection guards, but they also need to check the integrity of each other [28].

Software guards divide the program into three types of code regions: (1) *user code*, which is the program's original logic; (2) *checker code*, which checks the integrity of a code region by comparing its hash value to an expected value; and (3) *responder code*, which replaces a tampered code region with the original code for that region. Multiple checkers can check a region and multiple responders can replace a tampered region. The tamper resistance is provided by a network of guards placed in the execution flow graph of the program in such a way that: (a) the responders are inserted before the region they guard; (b) the checkers are inserted after the region they check is present in the program image. Increasing the number of guards in the program enhances the resilience of the program against tampering, but also increases the overall size of the program code and affects the performance of the program.

3.3 Related Work

Existing Web service standards address security extensions for Service Object Access Protocol (SOAP)-based services. WS-Security provides specifications for cre-

dential exchange, message integrity, and message confidentiality during service interactions. WS-Policy provides specifications for advertising policies of services and policy requirements of clients. These standards could be sufficient for point-to-point security and policy enforcement in static service invocations, but they fall short of the policy transmission and enforcement in dynamic service invocations.

Access control in Web services is well-studied. Security Assertion Markup Language (SAML) is an open-standard specification and a framework for exchanging authentication and authorization information between parties in XML [42]. It defines three parties: (a) the principal requesting a service, the Identity provider (IdP) authenticates the principal and issues identity assertions, and the service provider (SP) uses the assertions and authorization policies for making access control decisions. It has been used to provide single sign-on (SSO) in a single domain. The implementation uses browser cookies to maintain the authentication state information. However, cross-domain implementation is a problem because cookies cannot be transferred across different DNS. Domain specific proprietary implementations further lead to non-interoperable solutions. Shibboleth defines an architecture based on SAML and provides an open-source implementation that allows federated identity management, authentication, and authorization [43]. It enables cross-domain single sign-on. The solution is prone to the TTP related issues because of its dependence on the IdP.

XACML is a more recent and advanced effort for enforcing authorization policies across heterogeneous domains. We provide a detailed discussion of XACML in Section 3.2.2. The proposed framework uses XACML-based access control policies and the policy evaluation engine.

Several general approaches have been proposed for controlling access to shared data and protecting its privacy. Park et al. present a classification of security architectures for access control of sensitive data and secure data dissemination [44]. The classification is based on three elements: (a) the type and presence of a trusted monitoring and enforcement mechanism that controls and manages access and usage

of sensitive data; (b) the specification of access and usage policies; (c) the method of data dissemination. They assume that the trusted monitoring and enforcement mechanism is stationary. In contrast, the proposed framework uses a portable mechanism for controlling data disclosure.

Various solutions assume that data recipients are known in advance and propose cryptographic approaches to encrypt data in such a way that each recipient can decrypt only the data that it is allowed to access. For instance, Digibox is a cryptographically protected data container that uses multiple keys to enforce policies [45]. An entity must get a required decryption key to access a particular data item. A detailed discussion of cryptographic solutions is presented in Chapter 2 (Section 2.2.1).

DataSafe is a software-hardware architecture that supports data confidentiality throughout their lifecycle [46]. It is based on additional hardware and uses a trusted hypervisor to enforce policies, track data flow, and prevent data leakage. Applications running on the host are not required to be aware of DataSafe and can operate unmodified and access data transparently. The hosts without DataSafe can only access encrypted data, but it is unable to track data if they are disclosed to non-DataSafe hosts. The use of a special architecture limits the solution to well-known hosts that already have the required setup. It is not practical to assume that all hosts will have the required hardware and software components in a cross-domain service environment.

A privacy preserving information brokering (PPIB) system has been proposed for secure information access and sharing via an overlay network of brokers, coordinators, and a central authority (CA) [47]. This solution is based on the notion that the brokers may not be trusted and could collude to correlate, infer, or leak information. To provide protection, it proposes an approach to divide and allocate the responsibility and processing among multiple brokers so that no single component has enough control to make a meaningful inference from the information disclosed to it. However, the ownership, distribution, and management of components are a challenge in a large cross-domain system. The approach does not consider the heterogeneity of

components such as different trust levels and policy conflicts among them. It uses a centralized TTP to manage metadata, joining/leaving of brokers, and key management. The use of TTP creates a single point of trust and failure. The information can be leaked if the TTP is attacked and compromised.

Other solutions have been proposed that address secure data dissemination when the recipients are not known in advance. Pearson et al. present a case study of EnCoRe project that uses sticky policies to manage the privacy of shared data across different domains [48]. The main idea of sticky policies is to make data and policies inseparable so that an unauthorized recipient cannot access the data without satisfying the policies [49]. In the EnCoRe project, the sticky policies are enforced by a TTP and allow tracking of data dissemination. Other methods of implementing sticky policies include using Identity-based Encryption (IBE) or Trusted Platform Module (TPM). The approach is prone to TTP-related issues. The sticky policies are vulnerable to attacks from malicious recipients.

A TPM is a hardware component that can facilitate various security operations, such as authentication, hardware-based encryption, digital signing, secure key storage, and attestation of installed software [50]. TPM stores cryptographic keys that are not accessible to any other entity. It uses the internal keys to provide secure encryption and signing. It can also be used to detect software tampering by exploiting the chain of trust [51]. The solutions that use TPM are limited by the availability of the TPM in external domains. Moreover, the TPM does not ensure policy enforcement in any way, it only helps in assuring secure code execution.

The HP time vault service provides a solution for document dissemination to a set of recipients [52]. The document is encrypted using a symmetric key and the symmetric key is also encrypted using a key based on the future date and time of the document's disclosure. The time value service uses a trusted clock to generate and publish decryption keys associated with the current date and time. The encrypted symmetric key and the document are shared with the recipients. A recipient can only receive the decryption key to decrypt the symmetric key at the specified disclosure

time. The recipient can then use the decryption key to first decrypt the symmetric key and then use the symmetric key to decrypt the document. In this solution, the policies are limited to the timing conditions.

The bundle scheme provides an approach for protecting the privacy of disseminated data [53]. This approach is a precursor to the Active Bundle scheme used in the proposed framework. A bundle encapsulates the sensitive data and the associated policies. It trusts the recipient to enforce the bundle's policies which is not practical for cross-domain interactions.

3.4 Proposed Solution

In this section, we describe the proposed solution for policy transmission and enforcement to control data disclosure to services. Earlier, we presented the definition of a new solution and identified the main requirements for its correctness 2.3. In this section, we use that definition to model the proposed solution and discuss its realization.

The solution is based on the idea of execution monitoring [19] of the data access requests (described in Subsection 3.2.2). The Execution Monitor (EM) receives requests from services for data access and permits them to access the data only if the applicable policies allow it. It uses the history of execution steps and their outputs, such as a step for authentication of service and a step for authorization of data access request based on the evaluation of applicable policies, to decide whether the action is authorized and allows its execution, or the action is unauthorized, i.e., violates the applicable policies and terminates its execution. The execution steps such as authentication, authorization (policy evaluation) are controllable actions and, therefore, their execution can be terminated by the EM [36].

The access policies that we consider in this system are enforceable by execution monitoring as they satisfy the enforceability conditions [36]: (a) inspecting the execution steps of the service interaction is sufficient to determine whether it is policy

compliant; and (b) the execution steps of previous interactions are independent and do not affect the policy compliance of the current interaction. In relation to standard access control architectures such as XACML, the EM acts as both, a PDP that is used to evaluate data access requests against applicable policies, and a PEP that is used to intercept access requests and enforce policies by approving or denying them based on the evaluation results of the PDP.

In a service environment, assuming the enforcement mechanism E is present at the service, it works as follows. The system, in this case, is a service S that sends a data access request a (action) to the E. E intercepts the request a and checks whether the execution of the data access a is authorized, i.e., it does not violate the given policy P (a set of policies applicable to a). E then permits the execution of the data access request a and discloses the data to S or denies the request if the policy P is violated.

In a composite service environment, the client request (including the client's data) is disseminated to external service domains which are hidden and possibly unknown to the client. This adds a restriction on the placement of the EM because the EM needs to be available in every domain, where data is disseminated and access is requested, to ensure the enforcement of policies. The existing policy enforcement mechanisms place EM either at the source, the destination, or a mediator. Since the component services in the composition are dynamically decided and hidden from the client, the placement of the EM at the source or a mediator is not possible. A pre-defined placement of EM at all possible destinations (component services) is not feasible, especially in a cross-domain multi-provider environment with real-time constraints.

This chapter proposes a mobile EM that can be dynamically placed on the component services that participate in an interaction to control data disclosure and ensure end-to-end policy enforcement. The solution is realized using the AB approach described in Subsection 3.2.3. The AB embeds the sensitive data, the related policies as metadata, and the EM as the monitor. In this way, the EM travels along with the data and the policies as part of the AB. The client shares the data by means of an

AB that is included in the request to the composite service. The composite service forwards the AB to its component services to transmit client's data, policies, and the EM. The services interact with the AB to send requests for data access. The requests are intercepted by the AB's monitor (EM), which is responsible for enforcing the policies and allowing access to the data. We demonstrate that the proposed solution satisfies the real-time constraints required in the Web service environment.

3.5 Implementation

In this section, we present the implementation of the AB approach and its working in the framework. A client (data owner) uses the framework as follows to interact with the service providers. He provides the data, the applicable access policies, and the target service to the AB Generator (ABGen) application, that uses the information to generate an AB and sends the AB to the target service. The description of the implementation of Active Bundle, the ABGen, the ABHandler, and the utility composite Web service used for the experiment follows.

3.5.1 Active Bundle Design

The AB includes data, metadata, and a monitor, as discussed in Subsection 3.2.3. Note that the AB implementation, we describe, is for the stateless and immutable AB. However, the implementation could be easily extended for other modes and structures. The three components are implemented as follows:

Data: Each data item is encrypted and stored as key-datum pair using JavaScript Object Notation (JSON[2]) format, e.g. { "ab.email" : "E(abc@xyz.com)" }. This organization allows services to query AB for specific data items using the *key* attribute. The symmetric key for encryption is based on the AB execution control flow steps. More details are discussed in the Key Derivation Subsection 3.5.4.

[2]JSON is a lightweight data exchange format. http://json.org/.

Metadata: They include the access control and operational policies specified by the data owner. The operational policies are the operational requirements of AB, e.g. expiration time of AB. The access policies are rules for accessing the data. The operational policies are hard-coded in the AB Monitor code. The access policies are specified using the ABAC model—other access control models such as RBAC are also supported. The policies are implemented using eXtensible Access Control Markup Language (XACML) 3.0. An example policy for healthcare data expressed using XACML elements, is shown in Table 3.1. This policy allows access to the Electronic Healthcare Record (EHR) only if the access request comes from Dr. Alice's application and denies access in all other cases. However, in an emergency context, it also allows data access to the applications of paramedics and other doctors.

Table 3.1.
An example access policy for healthcare data.

DENY	
Resource	Electronic Healthcare Record
Subject	Dr. Alice
Action	Read
Environment (emergency context subject)	Paramedics, Doctors

Monitor: It is a set of Java classes that execute the AB, handle access requests, and perform policy evaluation and enforcement. The AB's monitor is modeled based on the required components identified in Chapter 2 (Section 2.3) to ensure the correctness of its operation. These components are discussed as follows:

- Host Authentication: It is based on the use of signed digital certificates. The services present their X.509 certificates signed by a trusted Certificate Authority (CA) to verify their authenticity to the AB. This is the first step during the interaction between an AB and a service.

- Data Access Authorization: It uses the ABAC policies that are designed to evaluate the attributes of the interacting services (e.g. trust level), the data item request, and the environment conditions (e.g. emergency context, attack context). The evaluation of applicable policies determines the data disclosure. The policy evaluation is implemented using an open source XACML PDP known as WSO2 Balana[3].

- Data Disclosure: It identifies the data item that needs to be decrypted. The decryption key is dynamically derived using the unique information generated during the AB's execution only if the service is authenticated and its data access request is authorized. More details are provided in Subsection 3.5.4.

- Tamper Resistance: It is based on the verification of the code of the AB's monitor. It uses a secure one-way hash function to calculate the digests of the different AB modules (Java classes) in the monitor and their resources and uses this information for key derivation. The correct keys are only derived if the code has no modification. More details are provided in Subsection 3.5.4.

3.5.2 AB Versions

There are four variations of AB implementation. These are described as follows:

ABx and ABxt

In these versions, the AB policies are specified in XML according to the XACML specifications. The WSO2 Balana library is used as a PDP for policy evaluation. The ABxt version includes tamper resistance functionality in addition.

[3]WSO2 Balana is an open source PDP for XACML. https://github.com/wso2/balana

ABc and ABct

In these versions, the AB policy elements (such as subject, object, resource, etc.) are specified in JSON and the PDP is implemented as conditional statements in Java code that evaluate the policies. The ABct version includes tamper resistance functionality in addition.

3.5.3 AB API

AB provides a set of API methods for interaction with services. The API is implemented using Apache Thrift framework.[4] The API methods are described as follows:

`getSLA()`:

AB promises to provide data/service with certain guarantees which are represented as AB's Service Level Agreements (SLAs). The SLAs are based on the operational policies specified by the data owner during AB creation. The SLA information is accessible without restriction through this method. Examples of this information include expiration time of AB, maximum time for which an AB will stay active during an interaction, etc.

`getValue(req, sign, cert)`:

Services call this method to request a data item from AB. Figure 3.6 shows the UML activity diagram of AB steps in response to a request. The method takes three parameters as follows. *req* is the key of the requested data item, e.g. *ab.email*. Service signs the *req* using their *privatekey* to generate signed request *sign*. *cert* is the certificate of the service. Service certificate should be signed by a trusted CA so that the AB can trust the service's authenticity. AB verifies the authenticity of the request as follows. It checks if the *req* is valid and signed by the service's *cert*. Then it verifies that the service's *cert* is valid, has not expired, and is signed by a trusted

[4]Thrift is a cross-language Remote Procedure Call framework that defines an interface language and a binary communication protocol to develop services that work seamlessly across different implementations. https://thrift.apache.org/

CA certificate. Next, the AB determines the applicable policies in the context of this access request and authorizes the request by evaluating the policies. The evaluation is based on the attributes (ABAC) of the access subject (service's certificate, trust), object (data being requested), and the environment conditions (such as emergency context). AB uses the unique information generated during its control flow steps (such as authentication/authorization results) along with the digests of the AB modules to derive the symmetric key that was used to encrypt the data item during AB creation. AB uses the key to decrypt the data item and returns it to the service.

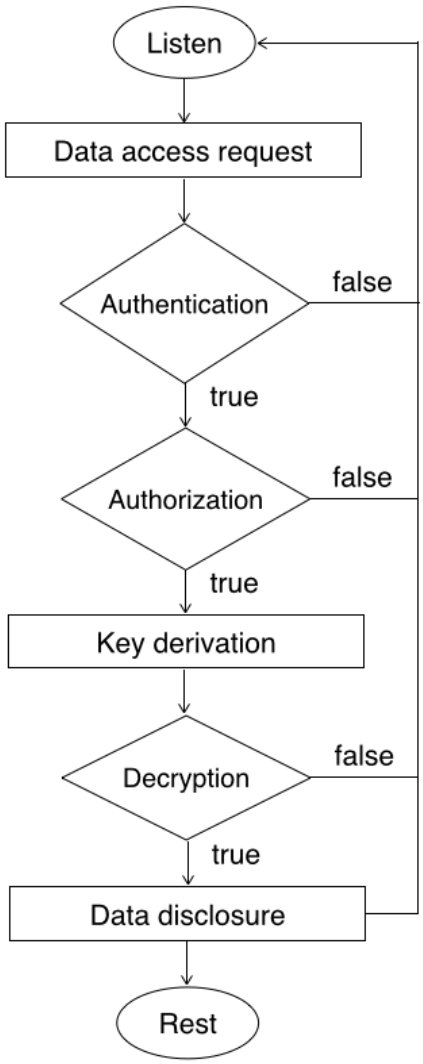

Fig. 3.6. Interaction of active bundle and service.

`getSecureValue(req, sign, cert)`:

This method performs all the functions of the `getValue(req, sign, cert)`. Additionally, it also encrypts the data item with the *public key* of the service (which is extracted from the service's certificate) before returning it to the service. The service uses its *private key* to decrypt the data item. This method helps in preventing attacks such as Man-in-the-Middle attack (MITM) attack.

3.5.4 AB Generator

It is implemented as a Node.js[5] Web application ABGen, which is used to create ABs. Data owners use the application interface to specify the data (as key-value pairs), applicable access control policies (as XACML elements), operational policies (as AB SLA parameters), the AB template, and the Web address of the target service for AB. An AB template defines the program skeleton of AB and is used to generate ABs with the same structure. It includes the implementation of the invariant parts (monitor) and placeholders for customized parts (data and policies). ABGen can use multiple AB templates, for instance, based on the different CA certificates.

The ABGen application parses the user input and processes it to derive the symmetric keys for data encryption. It encrypts the data and adds the cipher text and the policies to the specified AB template. Next, it generates the AB as an executable Java Archive (JAR[6]) file. It serializes the AB file using *Base64* encoding to preserve the data format and appends the output to the HTTP body of the message, which is sent to the specified target service. Figure 3.7 shows the creation of an AB by ABGen based on the data and policies of a user. AB is then included in the request message and sent to the specified service.

[5]Node.js is an open source platform for easily building scalable, data-intensive, and real-time applications. https://nodejs.org/

[6]JAR is a standard way of packaging Java classes, metadata and resources into one file. It helps to distribute Java code.

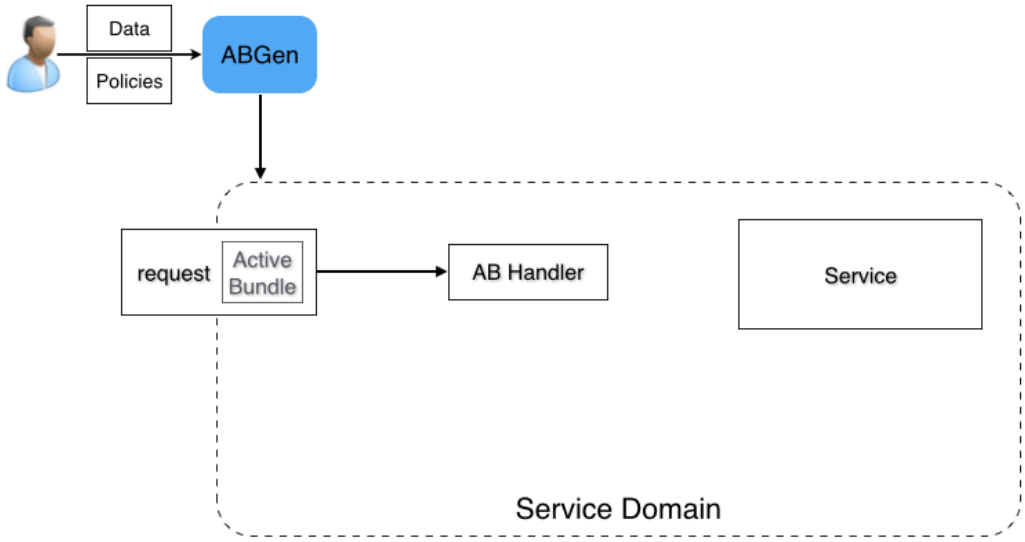

Fig. 3.7. Creation of active bundle.

Key Derivation

The symmetric keys for AB data items are derived based on the unique information generated in the execution control flow path of AB. The main control flow steps include authentication and authorization. These steps generate unique information during an interaction with a service only if the service is authentic and its request is authorized. A Key Derivation Function (KDF[7]) is used to derive the key based on the information. In order to ensure proper entropy in the key, the information is first transformed into a secret by taking the hash of the information; the secret is then used to derive the key, using methods such as *SecretKeyFactory*, *PBEKeySpec*, and *SecretKeySpec* provided by *javax.crypto* library. During AB creation, the policies of the user are first embedded in the AB template. AB template is then executed to obtain the information and derive the related keys for each data item. Each data item is then encrypted using the related key. An example of the distinct information for each key is the set of authorization policies applicable to the associated data item. During interaction with an authenticated-authorized service, the AB execution control

[7]A KDF is a deterministic algorithm for deriving cryptographically strong secret keys from some secret value [55].

flow steps generate the same information, which is used to derive the appropriate key and decrypt the data item.

Tamper Resistance

The correctness of data dissemination depends on the correct execution of AB control flow steps. Tamper resistance checks the integrity of the AB execution to ensure it has no difference from the original code. Each time an AB step is executed, the tamper resistance module calculates its digest and the digest of the resources used by it (e.g. authorization module and the policies that it evaluates) through a secure one-way hash function (such as *SHA-2*). These digests are combined with the information used for key derivation. Therefore, if there is any modification to any step or its resources, the digests change, and the derived key (based on the digests) is incorrect.

3.5.5 AB Service Handler

It is a Web service middleware extension, implemented as a Node.js module that intercepts incoming requests before being serviced by the Web service APIs. Figure 3.8 shows an AB Handler in a service domain that intercepts an incoming request with an AB.

The handler is invoked for each service request and performs the following steps:

- It checks and proceeds only if the incoming request includes an AB otherwise it passes the request to the next middleware extension.
- It extracts the AB from the request.
- It decodes the AB and stores the AB file on the filesystem.
- It generates a port number and starts the AB on that port.

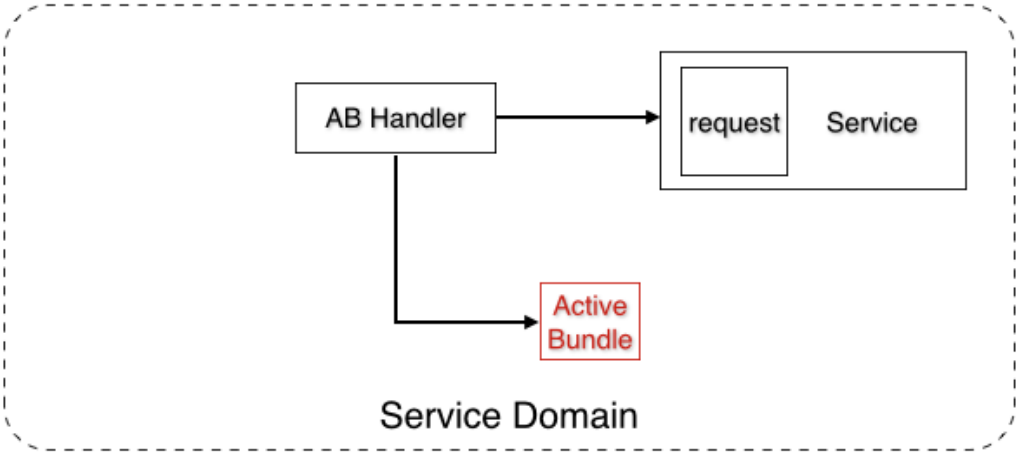

Fig. 3.8. Extraction of active bundle.

- It forwards the request to the service and sends the port number of the related AB.

The AB starts execution on the specified port and is ready for interaction with the service. The service receives the request and interacts with the AB process on the specified port to send a data access request. Figure 3.9 shows an interaction between the AB and the service.

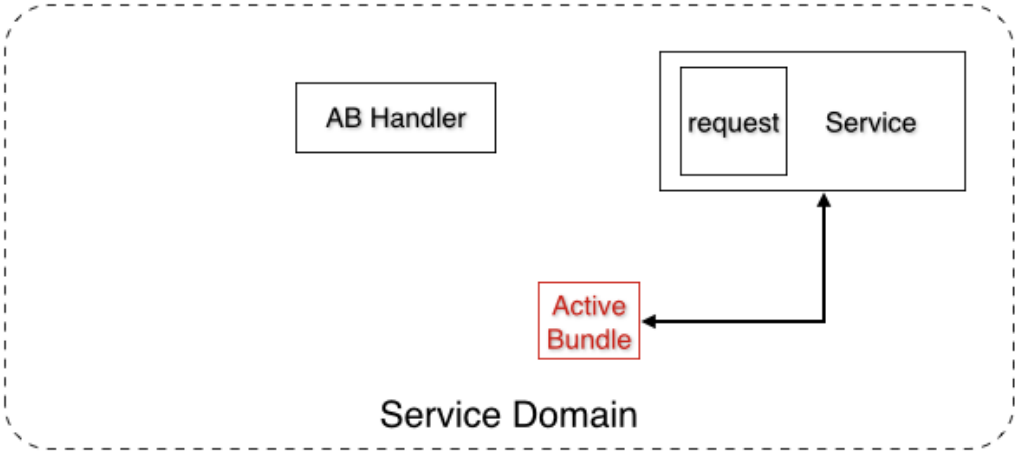

Fig. 3.9. Execution of active bundle.

3.6 Demonstration Scenario

In this section, we present an online-shopping scenario and describe the application of the EPICS framework. The service requests that include sensitive data are conducted using ABs. An AB is created based on the client's data and policies and sent to the service provider. In order to interact with the component services, the service provider forwards the AB to the services. AB interacts with the services on behalf of the client and disseminates data based on the policies associated with requested data.

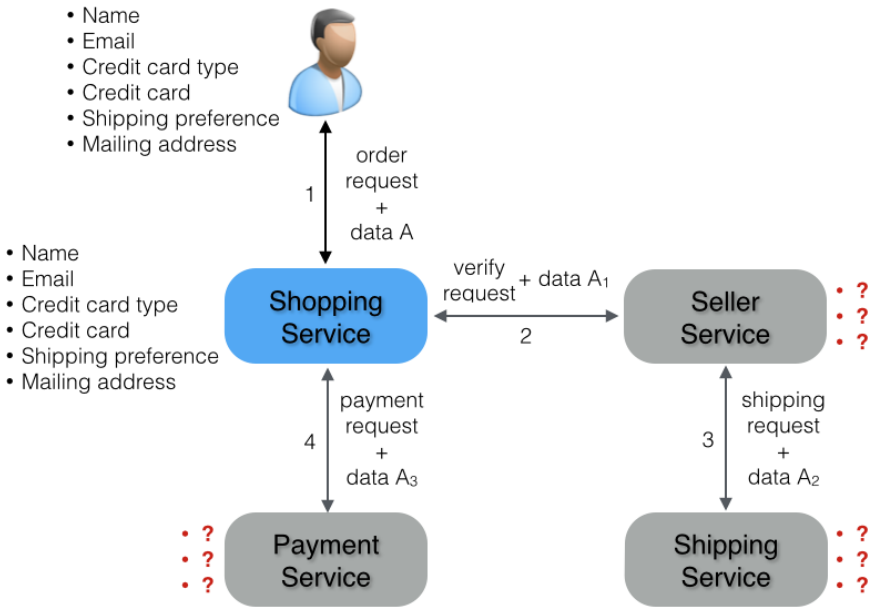

Fig. 3.10. Composite Web service for online-shopping.

3.6.1 Conventional Workflow

Consider a composite Web service for online-shopping as shown in Figure 3.10, for e.g., e-commerce services offered by Amazon or E-bay. The user initially registers with the shopping service by creating an account. During registration, the user has to disclose sensitive information to the shopping service including name, email, credit

card, mailing address, billing address, phone number, etc. The user sends an order request to the shopping service, which communicates with the seller service to verify the order (e.g. to check item availability in the specified quantities, sizes, colors). The seller service then shares the information with the shipping service to verify shipping eligibility. On verification from the seller service, the shopping service applies tax and shipping charges and calculates the total amount due for the order. User information is sent to the respective payment service for verification. On payment approval, the order is completed and the client and the seller service are informed.

Shopping service does not require all the information of the user in order to provide its service to the user. For instance, the service can charge to the user's credit card for an order without knowing it. Services often justify by arguing that the stored information is encrypted, but if the service is compromised, user information is leaked. Service data breaches are becoming commonplace and have been well documented. For instance, attackers were able to get access to the sensitive user information such as credit cards, mailing addresses, email addresses, phone numbers, birth dates, medical IDs, social security numbers, and employment information during the Target Data Breach [1] and the Anthem Data Breach [2]. Furthermore, the user, the owner of the information has no visibility or control over how their information is used or shared. In this scenario, the information is shared with the seller service, payment service, and shipping service, but these interactions are not visible to or authorized by the user. Such interactions may disclose user data to unauthorized parties (from the user's perspective) and the user has no means of detecting such data leakage.

The main observation in this scenario is that there are multiple services, each with a separate functionality. Thus, each service requires different data to provide its service. For instance, the shipping service only needs the address of the user, the payment service only needs the payment credentials, and the shopping service needs to authenticate the user. No single service needs all the information, but online-shopping services currently store all the information associated with a user.

3.6.2 EPICS Workflow

The goal of the proposed framework is to selectively disclose information based on the policies, minimize unnecessary disclosure, and ensure security and privacy of the information. Our current solution uses the AB to achieve this. The use of AB ensures that the security policies are transmitted and enforced during data dissemination.

We assume that the client and the shopping service have a pre-existing mutual trust relationship, for instance, established during client's registration with the shopping service. During registration, the client, instead of sending all their information, sends an AB (which includes the information) to the service. The shopping service stores the AB and associates it with the client's account. When the client logs in, the shopping service starts the AB and gets only the information from the AB which it requires to provide the service. When the client sends an order request, the shopping service sends the AB along with the requests to the seller and the payment services. Similarly, the seller service sends the AB along with the request to the shipping service. Each service interacts with the AB and gets only the information which it needs to provide the service. After completing the request, the seller, shipping, and payment services discard the information and the AB. The shopping service stops the AB when the client logs out and discards any information received from the AB. This reduces the service liability to the client because if the service is compromised, the client information is not leaked and the attacker only gets access to the client's AB.

3.6.3 Scenario Implementation

The scenario includes the composite service (shopping) and the three component services (seller, shipping, and payment). Clients interact with the shopping service to send order requests and share sensitive data by means of AB. All services receive the AB, execute it, and interact with it to provide their service. We implemented the services as Node.js Web applications in Javascript. These services are loosely-coupled

and provide RESTful APIs for interaction. Services communicate over the standard HTTP protocol to receive requests and send responses.

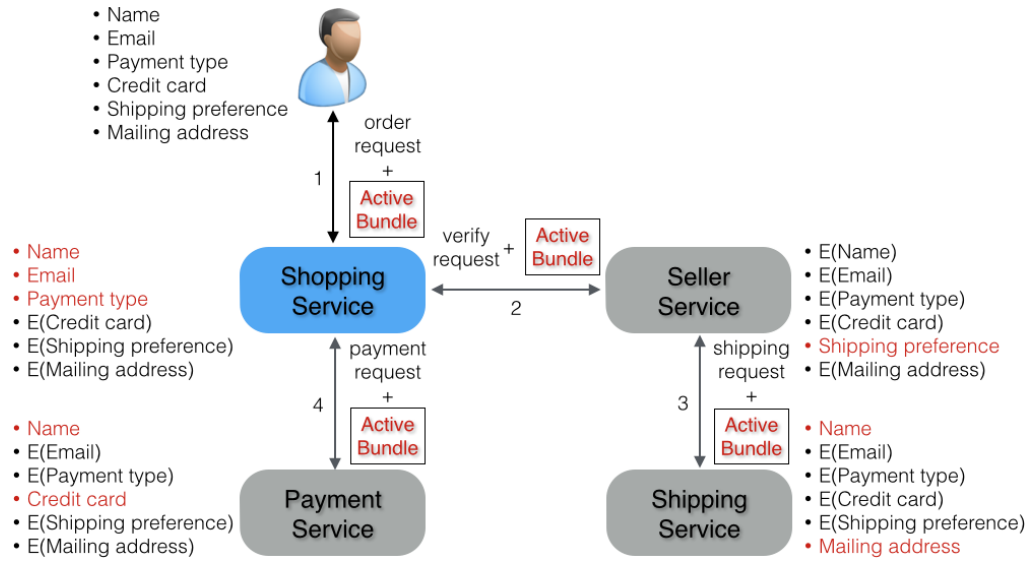

Fig. 3.11. Online-shopping scenario using EPICS.

3.7 Evaluation and Experiments

The system is evaluated for performance and security in the context of the online-shopping scenario. Figure 3.11 shows the updated scenario based on the proposed solution. The user creates an AB and sends it along with the order request to the shopping service. The shopping service sends requests to other services in the composition and forwards the AB along with the requests. In this case, the services interact with the AB, which has all the information, but only get access to the information for which they are authorized. For instance, payment service only gets access to the name and credit card data of the client and the shipping service only gets access to the name and mailing address data of the client.

3.7.1 Performance

We conducted a series of experiments to measure the overhead of using the AB for interaction with the services. We describe the experimental setup and present the results in the following sections.

Experimental Setup

The experiments were conducted using the Amazon EC2 cloud[8] to benchmark performance. The variables used in the experiment are described as follows:

1. AB versions: These include four different implementations of AB — ABx, ABxt, ABc, and ABct (described in 3.5.2).

2. AB policies: The number of applicable policies for a data item was varied exponentially (2^x) from 1 to 16.

3. AB execution environment: Two different execution environments were used — EC2 Large instances (CPU: 2, Memory: 3.75 GB, SSD) and EC2 XLarge instances (CPU: 4, Memory: 7.5 GB, SSD).

The AB is created with six data items (name, email, credit card type, credit card, shipping preference, mailing address), as shown in Figure 3.11, and at least one applicable policy for each data item. Example policies for credit card, credit card type, shipping preference, and mailing address are shown in Tables 3.2, 3.3, 3.4, 3.5 respectively. The policies define the applicable resource (data item), the allowed subject (service), the permitted action (read access), and the acceptable environment (service rating). An interaction between the AB and a service involves — service sending access request to the AB for a data item, AB authenticating the service, AB authorizing the service request for the data item using applicable policies, AB decrypting the data item, and AB sending the response back to the service. It may

[8]We used the EC2's C3 instances for experiments as they are suitable for Web-servers. http://aws.amazon.com/ec2/instance-types/

also involve integrity verification (tamper resistant code evaluation) when ABxt or ABct versions are used.

Table 3.2.
Policy: Payment.

DENY	
Resource	Credit card
Subject	Visa payment services
Action	Read
Environment	Rating <4

Table 3.3.
Policy: Payment type.

DENY	
Resource	Credit card type
Subject	Shopping service
Action	Read

Table 3.4.
Policy: Shipping preference.

DENY	
Resource	Shipping preference
Subject	Seller service
Action	Read
Environment	Rating <4

Table 3.5.
Policy: Mailing address.

DENY	
Resource	Mailing address
Subject	Shipping service
Action	Read
Environment	Rating <4

The online-shopping scenario setup includes the four services — shopping, seller, shipping, and payment. The shopping service uses the seller service to verify and place the order. The seller service uses a shipping service to ship the order. The shopping service uses a payment service to charge the client. Under the EPICS workflow, the shopping service already has the client's AB (supplied by the client during registration). In the experiment, the client sends an order request to the shopping service, which selects the appropriate seller and forwards the client's AB along with the request. The seller service, after order verification, sends the AB to the shipping service along with the request. On receiving shipping validation from the shipping service, the seller verifies the order to the shopping service. The shopping service, then forwards the AB to the appropriate payment service along with the request. The payment service verifies the payment status to the shopping service and the shopping service sends the completed order information back to the client. Each service interacts with the AB and requests only the data items for which they are authorized and need to provide the service.

Results

The results are reported based on the mean of data collected over 5 runs; each run included 100 interactions.

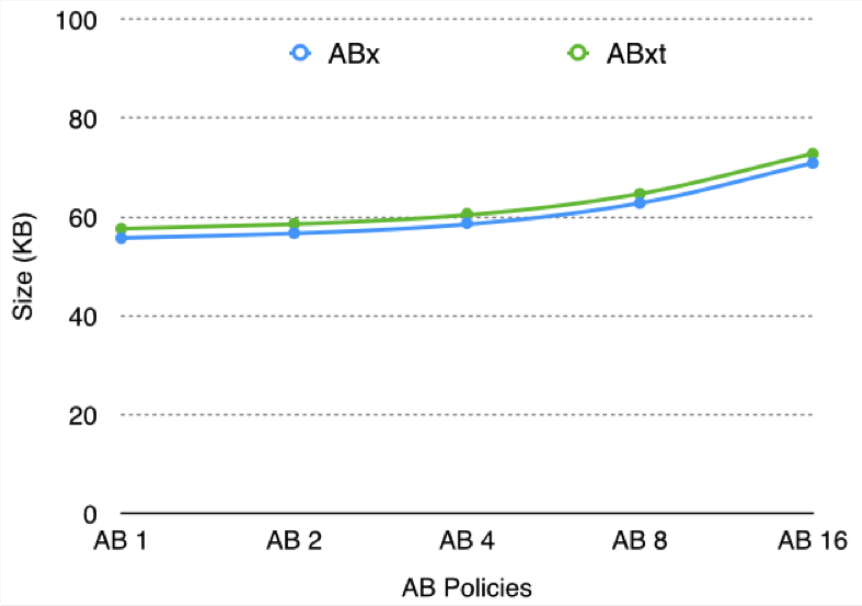

Fig. 3.12. AB size for ABx and ABxt.

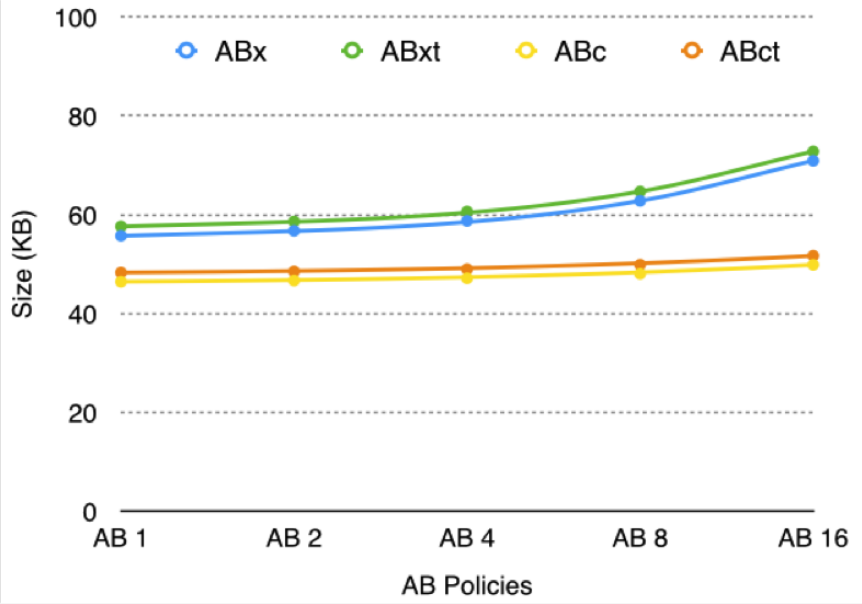

Fig. 3.13. AB size for ABc and ABct.

Figure 3.12 shows the graph for the increase in the size of ABx and ABxt versions with the increase in the number of AB policies. The results show that the use of

tamper resistance adds a slight overhead (< 2 KB) to the size of ABxt. When the AB implementation is switched to the JSON-based policies, the size of the AB is reduced. Figure 3.13 shows the graph for the increase in the size of ABc and ABct versions with the increase in the number of policies in the AB and their comparison with the ABx and ABxt versions. The use of JSON-based policies reduce the size of a policy by 78% (0.79 KB of reduction per XACML policy of size 1.01 KB) which leads to a significant reduction in the overall size of AB when multiple policies are used. There is an additional one-time reduction of 8.5 KB with the use of a Java-based policy engine with the JSON-based policies. The increase in size due to the use of tamper resistance in the ABct version follows the same trend as the ABxt version.

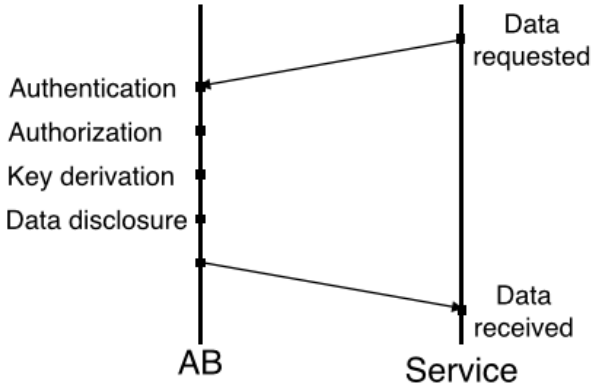

Fig. 3.14. Round trip interaction between AB and service.

Experiments were conducted to measure the interaction time between the AB and a service for a data item request under different variations. These experiments measure the round trip time to send the request to AB for data access, processing of the request by AB, and sending the response back to the service (as shown in Figure 3.14). The AB processing includes the authentication, authorization (based on policy selection, evaluation, and enforcement), and data disclosure.

Figure 3.15 shows the graph for the interaction time between AB and service on EC2 Large for ABx and ABxt versions with the increase in the number of policies in AB. The results show that the interaction time increases with the increase in

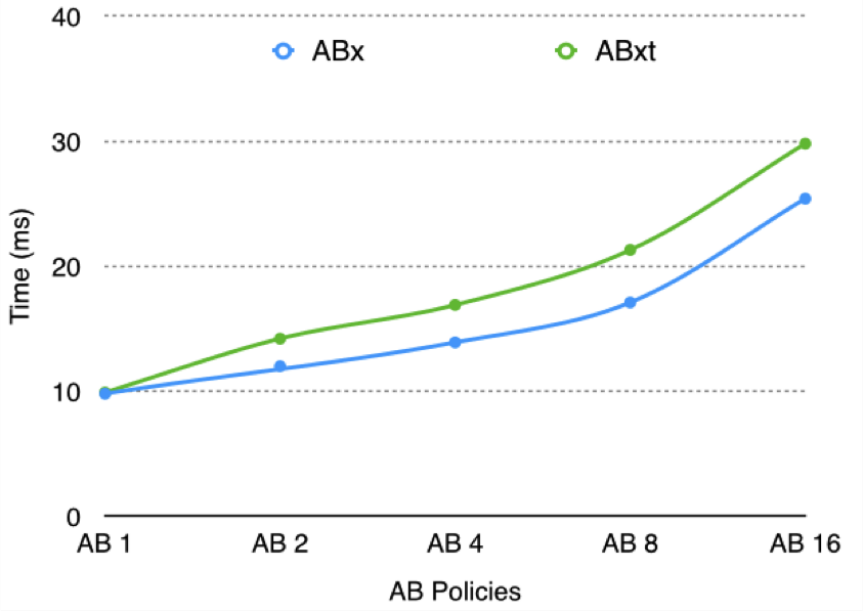

Fig. 3.15. ABx and ABxt interaction time with service on EC2 Large.

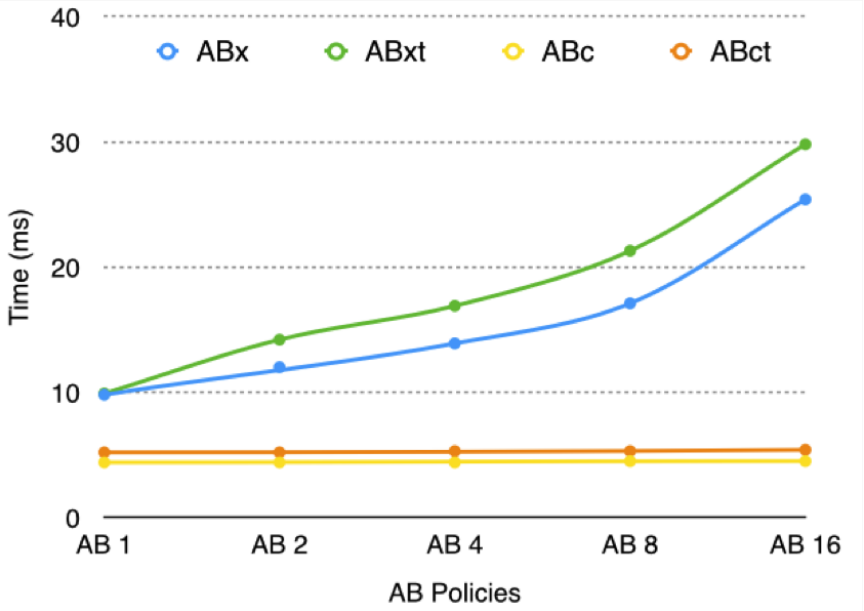

Fig. 3.16. ABc and ABct interaction time with service on EC2 Large.

the number of policies, but it stays under 30 ms for both versions with 16 policies. Figure 3.16 shows the graph for the interaction time between AB and service on EC2

Large for ABc and ABct versions with the increase in the number of policies in AB and their comparison with the ABx and ABxt versions. The use of ABc and ABct versions significantly reduces the interaction time. The difference in interaction time and growth is due to the difference in the implementation of policies. In ABc and ABct versions, the policies are specified in JSON and evaluated using conditional statements in Java code. They do not require any external library for evaluation, whereas, in ABx and ABxt versions, the policies are specified in XML according to the XACML specifications and the WSO2 Balana is used for policy evaluation. Evaluation of Java conditional statements is highly optimized, whereas the evaluation of XACML policies involves the traversal of XML policy and request trees which takes more time.

Figure 3.17 show the interaction time between the AB and a service on EC2 XLarge for all versions of AB. The graph confirms the earlier trends of interaction time on EC2 Large.

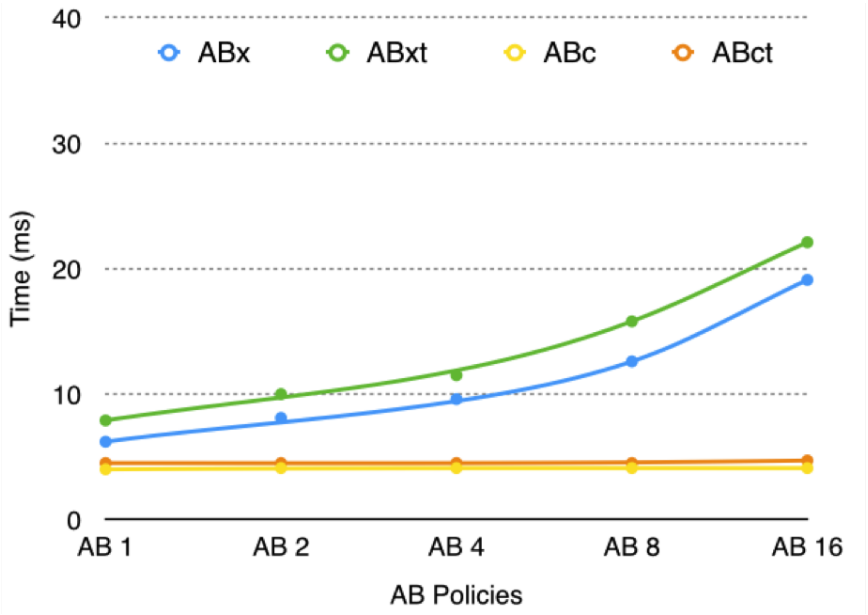

Fig. 3.17. AB-Service interaction time on EC2 XLarge.

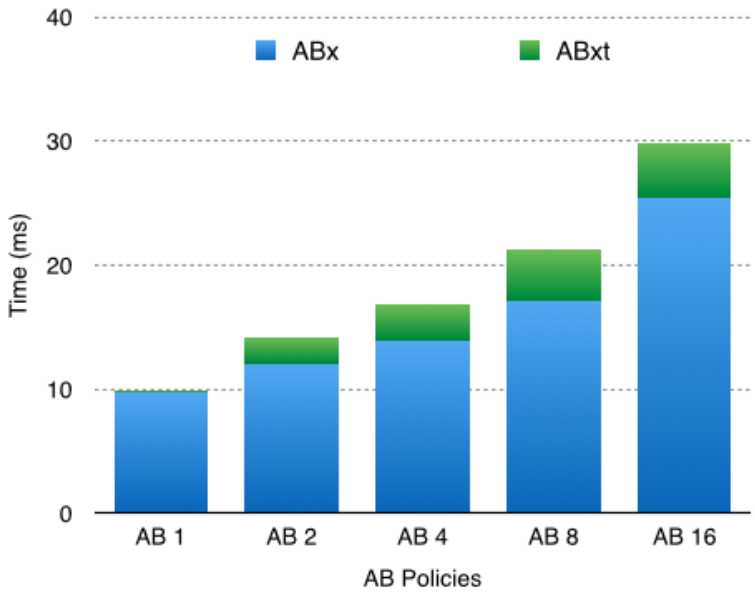

Fig. 3.18. Tamper resistance overhead with XACML on EC2 Large.

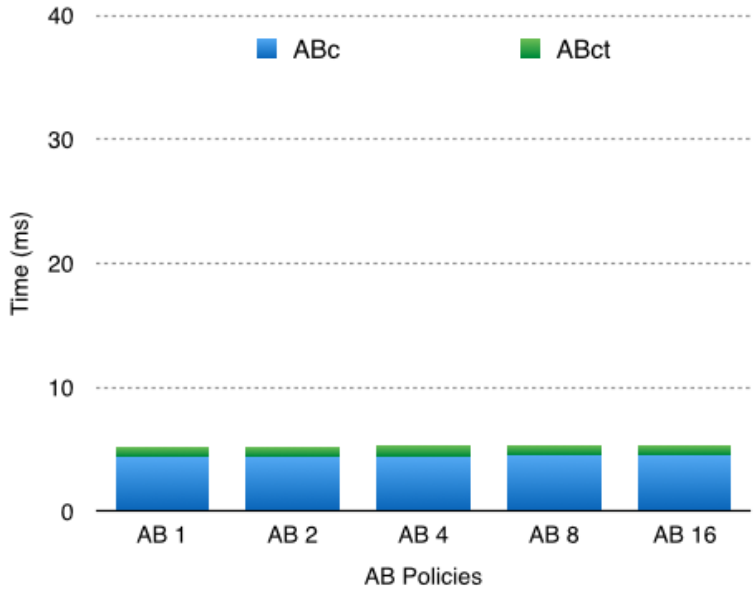

Fig. 3.19. Tamper resistance overhead without XACML on EC2 Large.

Figures 3.18 and 3.19 show the overhead of using Tamper Resistance in AB for the AB version that uses XACML policies and the AB version that uses JSON policies

respectively on EC2 Large. The reduction in overhead is achieved using JSON policies. The reduction comes from the tamper resistance's digest calculation of JSON policies which have a significantly smaller size than the XACML policies. Another factor that enables this reduction is the digest calculation of the authorization module, which uses the Java conditional statements for evaluation of JSON policies.

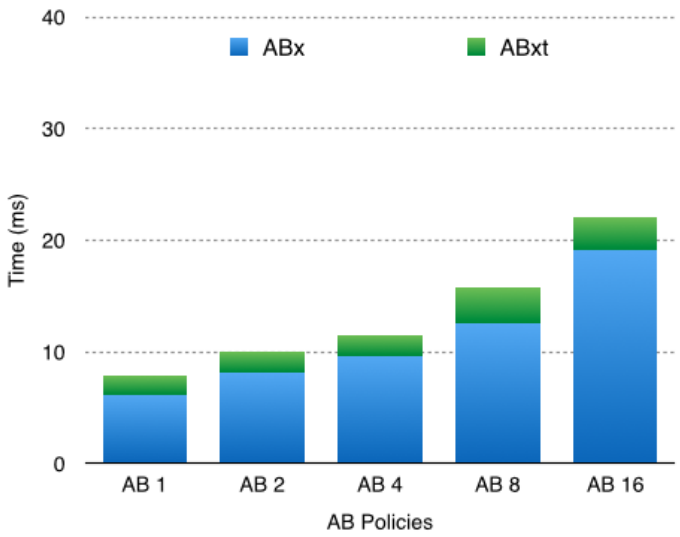

Fig. 3.20. Tamper resistance overhead with XACML on EC2 XLarge.

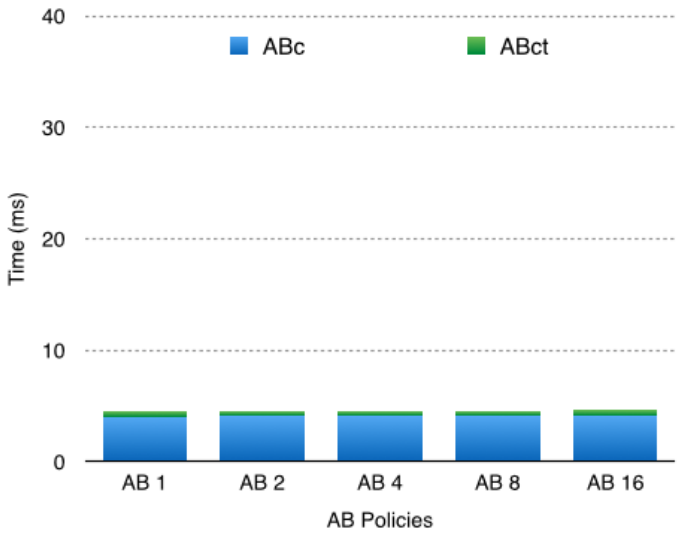

Fig. 3.21. Tamper resistance overhead without XACML on EC2 XLarge.

Figures 3.20 and 3.21 show the graphs for the overhead of using Tamper Resistance in AB for the AB version that uses XACML policies and the AB version that uses JSON policies respectively on EC2 XLarge. The graphs confirm the earlier trends of tamper resistance overhead for different AB versions on EC2 Large.

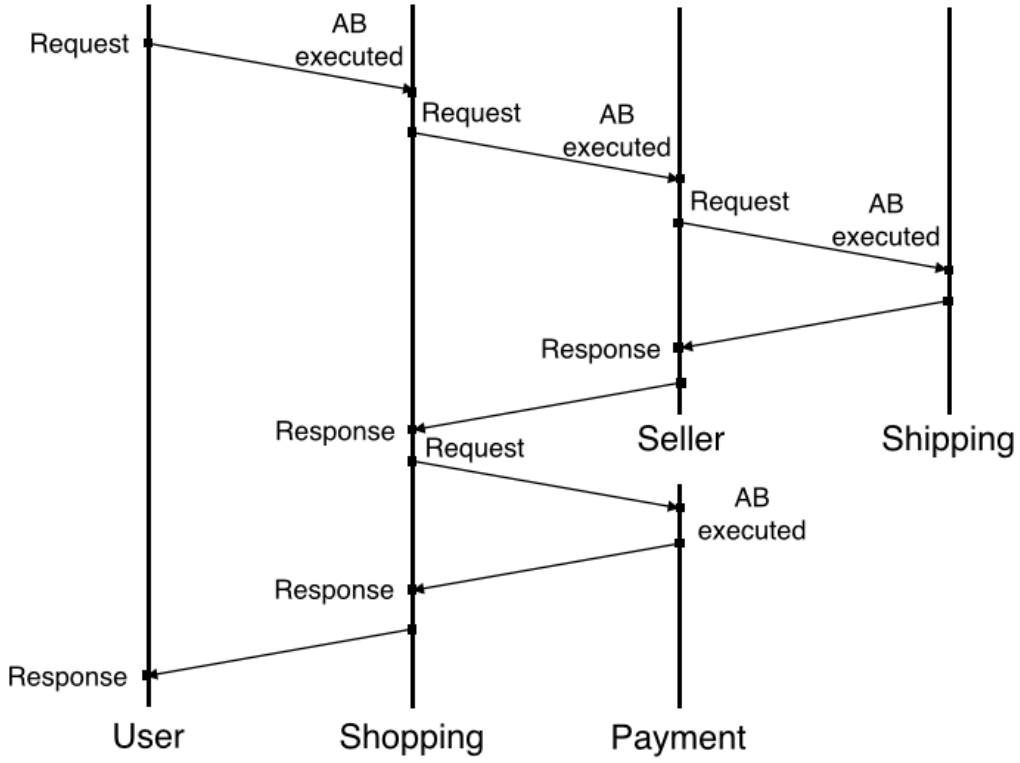

Fig. 3.22. Round trip interaction between client and composite service.

Experiments were conducted to measure the complete interaction time of the online-shopping scenario. Figure 3.22 shows the round trip interaction in the scenario. These experiments measure the round-trip time taken by the client's order request. It includes the network time to transfer the AB to the services in the composition, interaction time of the AB with each service, and the response time of each service.

Figures 3.23 and 3.24 show the results obtained for average round-trip scenario interaction time on EC2 Large and EC2 XLarge. The graphs follow the same trends as the interaction time graphs shown in Figures 3.16 and 3.17. The results show that the interaction time of the scenario is under 1.7 secs on EC2 Large and under 1.3 secs

on EC2 XLarge even with 16 XACML policies. The use of JSON policies reduces it to under 1 sec on EC2 Large and reduces it further on EC2 XLarge. These scenario times easily meet the real-time Web service constraints.

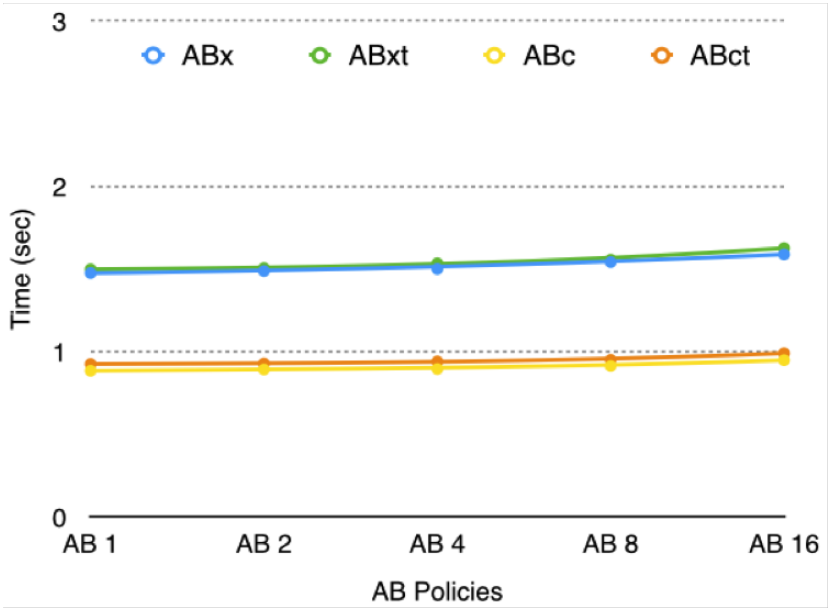

Fig. 3.23. Scenario time on EC2 Large.

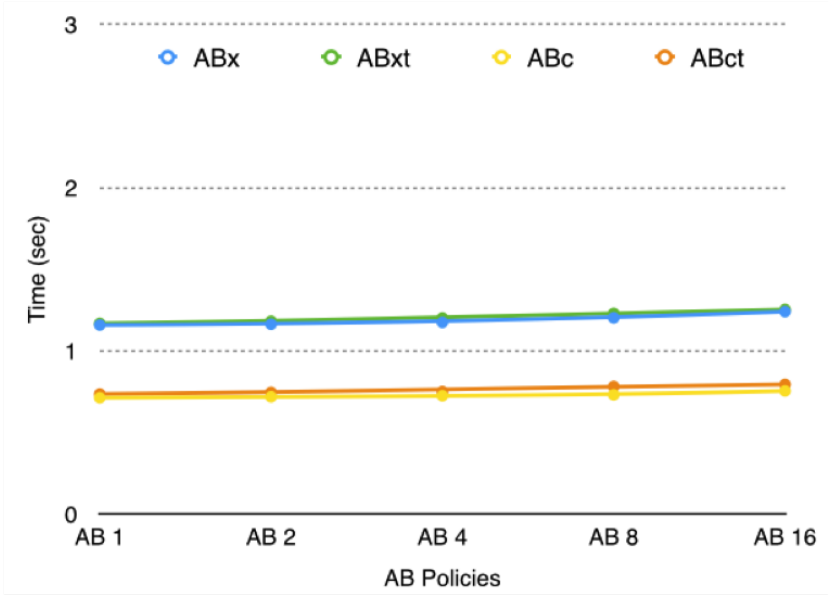

Fig. 3.24. Scenario time on EC2 XLarge.

3.7.2 Security

The framework is evaluated for security under different contexts. The threat model for the framework follows.

Threat Model

The privacy of the sensitive data shared by a client with a composite service can be compromised in several ways. The primary service selected by the client is generally trusted because a client would not want to use an untrusted service and share its data. The client establishes trust with the primary service based on mutual authentication-authorization or some pre-existing relationships. In the traditional model, once the trust is established, the client's trust boundary includes the primary service. When the primary service shares the client's data with other services in the composition, it extends the trust boundary to include these services. Note that the primary service also has pre-existing or dynamically established trust relationships with the component services. These services can be "honest-but-curious" and, therefore, interested in knowing more information than they are authorized to receive. For instance, services store client information to provide value-added services and analyze client's data to fine-tune their marketing strategies. This is a strong assumption because the services may be trusted, but may not be authorized to receive the entire set of the client's data. Note that a trusted service may leak data after an authorized access if it is compromised or turns malicious. This is an unavoidable situation analogous to an entity breaking another entity's trust.

In order to explain the threat model, we first define the adversary. An adversary is a service that receives the data and the associated policies of a client but ignores the enforcement of the policies. The Web service standards such as WS-policy [56] trust the EM of a service to correctly evaluate and enforce the policies. This is a very strong assumption that reduces the security of the data. The service may ignore the evaluation and/or enforcement of the policies. Furthermore, the service may

disseminate data to other services in the composition and may ignore to transmit the policies. The services that receive the disseminated data may not be authorized and being unaware of the client's policies may take further unauthorized actions on the data. In contrast, the proposed framework relaxes these assumptions. It requires only the AB to include a trusted EM, not each service. It provides a mechanism that enables the transmission of client's policies, along with the data, to each endpoint in an interaction and ensures evaluation and enforcement of policies to provide privacy-aware data disclosure. The goal of the EPICS framework is to provide protection against the following threats that are possible in the traditional model:

1. *Unauthorized data disclosure*: This threat occurs when an unauthorized service accesses the client's data or an authorized service accesses the data items for which it is not authorized. In EPICS, the use of policy evaluation and enforcement for each data access request ensures that the appropriate data is disclosed only to an authorized service.

2. *Ignored policy transmission*: This threat occurs when a service ignores to transmit the policies during data dissemination. In EPICS, the policies are always transmitted along with the data by means of AB.

3. *Disregarded policy evaluation*: This threat occurs when a service disregards the evaluation of the policies associated with the data. In EPICS, the use of EM (as part of AB) ensures that any request to access data is intercepted and evaluated against the applicable policies.

4. *Circumvented policy enforcement*: This threat occurs when a service accesses the data either by disregarding the policy evaluation or by ignoring the decision of the policy evaluation. In EPICS, the use of EM (as part of AB) ensures that the decision to allow or deny access to data is based on the results of the policy evaluation.

5. *Unauthorized data dissemination*: This threat occurs when a service disseminates the data to an unknown/unauthorized domain and the data is accessed by unauthorized services. In EPICS, the use of AB to transmit data and policies ensures that the services in any domain are unable to access the data if they are not authorized.

Other types of threats such as denial-of-service (DoS) attacks, data inferences, sybil attacks, masquerade attacks, side-channel attacks, incorrect data usage, compromised keys/certificates, etc. are beyond the scope of this work. In the following, we list the type of threats possible on the framework.

1. *Masquerade attack on service*: An attacker can impersonate a trusted client and create a malicious AB and send it to services within spoofed requests. A malicious AB can be used to launch attacks to steal information from the service or compromise the service.

2. *Masquerade attack on AB*: An attacker can impersonate a trusted service and send spoofed requests to an AB to gain unauthorized access to the data.

3. *Man-in-the-middle attack during AB transfer*: An attacker can intercept the REST message and get access to the AB during transfer. It can attack the AB to gain unauthorized access to the data.

4. *Man-in-the-middle attack during AB-service interaction*: An attacker can eavesdrop and intercept the communication between AB and a service. It can inject new messages to alter the communication and gain unauthorized access to the data.

5. *Tamper attack*: If an attacker gets access to an AB, it can compromise the AB by modifying its code and policies in order to attack a service or gain unauthorized access to the data.

6. *Execution hijack attack*: An attacker can reverse engineer and hijack the control flow of AB during its execution and modify it to gain unauthorized access to the data.

Resilience to Threats

There are two main aspects of resilience that address the above-mentioned threats. These are discussed as follows:

Trusting the AB

Services that receive an AB interact with it to access the data. They have to trust the AB and its execution and the data disclosed by the AB. An AB sent by a malicious party or a compromised AB can be used to attack the services. Therefore, the services need to ensure that the AB they receive is trustworthy before they execute it. This can be achieved as follows:

- *Digital signatures*: This approach is used to validate the authenticity and integrity of an AB. The sender digitally signs the AB with its secret key and the receiver uses the public key of the sender to verify the signature. A positive verification validates the source of the AB and the AB's integrity to ensure that it was not modified during transfer. It provides protection against Masquerade attacks on services and Man-in-the-middle attacks during AB transfer.

- *Execution isolation*: It is used to isolate the execution of an AB process in a service environment to protect the service from any malicious actions of the AB. The service can execute the AB in a containerized environment, for e.g., using Docker containers [57]. The containers separate the execution of the AB process from the underlying infrastructure and limit the actions of the process to the container. Thus, the infrastructure has limited exposure to the malicious AB process and cannot be easily exploited.

- *Cloud-based execution*: It is used to execute an AB on a third-party cloud platform. Cloud services offer Platform-as-a-Service (PaaS) model (e.g. IBM Bluemix [58], Google App Engine [59]), which can be used to execute applications on the cloud platform. Services can use the cloud platform to execute the AB and interact with it in the cloud to access the data. This prevents the AB's execution in the service domain and protects against the malicious or compromised ABs.

Protecting the AB

A malicious service that receives an AB can create a copy before executing the AB. The service cannot access the encrypted data in the AB without the decryption keys. However, it can attack the AB to gain unauthorized access to the data or compromise the AB to disseminate incorrect data or attack other services. Therefore, an AB needs to be protected against services that are malicious or have been compromised. This can be achieved as follows:

- *Secure communication*: An AB should be transferred using secure communication, e.g. HTTPS. This prevents Man-in-the-middle attacks during AB transfer. When a service interacts with an AB, the AB employs authentication to identify the service and verify the authenticity of the service to ensure that it is interacting with a legitimate service. This prevents Masquerade attacks on an AB. The AB API provides the `getSecureValue()` method, which a service can use to receive the data encrypted with its public key. The data can be decrypted only by using the secret key of the service. This prevents the Man-in-the-middle attacks on the interaction between an AB and a service.

- *Tamper resistance*: The AB uses code integrity checks to provide protection against tamper attacks. It uses the digest values of its modules to dynamically derive the secret keys for data decryption. The correct keys are generated only if the modules are unmodified, which ensures correct execution. This prevents

attackers from gaining access to AB's data (e.g. using Tamper attacks that modify the policies of the AB).

- *Hijack Prevention*: The main challenge in implementing the AB is assuring that a service executes the AB's code faithfully and correctly. The AB's monitor is written in Java, which is a type-safe language. This prevents the AB's execution control flow from several attacks such as Buffer overflow, illegal arithmetic operations, stack smashing, format string, etc. The AB code (JAR file) is digitally signed, which calculates the checksum of each file in the JAR and signs each checksum. The Java runtime environment validates the signatures and prevents the execution of malicious code if the signatures do not match. However, the AB implementation does not protect from the reverse engineering and side-channel attacks. For instance, a service may cheat by accessing the memory used by the AB to steal the decryption keys and access the data that the associated policies do not authorize the service to access. The protection against such attacks is possible if the service platform has a special purpose hardware or memory. For instance, techniques such as write protected memory can be used to prevent an attacker from modifying the memory where an AB is executing. The presence of a special purpose hardware, e.g. a TPM [51] can be used to execute sensitive operations of AB (e.g. authentication, hash calculation, data decryption) in isolation. Anti-Reverse engineering approaches such as HARES [60] employ Translation Lookaside Buffer (TLB) splitting that can be used to store the encrypted AB in the secure ("instructions") portion of memory, such that it is only decrypted during the execution with a key that resides in the processor. This prevents attackers from gaining access to the memory where AB's data and code resides during execution.

- *Cloud-based execution*: A trusted cloud platform can be used to execute the AB (e.g. IBM Bluemix [58], Google App Engine [59]). Services can interact with the AB executing in the cloud and get access to the data only if they are

authorized. Note that the cloud platform is used only for code execution; it does not broker data requests or perform policy enforcement and, therefore, does not get access to data. The execution of the AB in the cloud prevents tamper and hijack attacks.

The framework provides three levels of protection for data disclosure. The first level of protection is based on the authentication. Each service needs to authenticate with the AB in order to access the data. The second level of protection is based on the authorization. Each data access request is authorized based on the evaluation of the applicable policies. The third level of protection relies on the integrity-based key derivation. The data decryption is possible only if the AB is unmodified. The decryption discloses only the portion of the data for which the service is authorized.

3.8 Cost-Benefit Analysis of the Framework

The main benefits of using the EPICS framework are as follows:

- It ensures policy-based access control of a client's data based on the enforcement of policies.
- It provides privacy-preserving controlled data dissemination by minimizing the data disclosure.
- It provides context-based adaptable data dissemination based on the use of external environment information (such as trust values, an emergency or an attack context) and the flexibility of the policies.
- It is independent of TTPs and does not require the availability of the data owner to disseminate data to other services once the data is shared with the primary service.
- It reduces the liability of a service for managing a client's data by disclosing to the service only the data authorized by the client's policies.

- It is compatible with the existing service infrastructure such as HTTP-based RESTful services.

- It can be incorporated in any data dissemination application as it is policy language agnostic and supports a wide range of standard authentication and authorization mechanisms.

The costs related to the use of the EPICS framework and their justifications are as follows:

- There is an increase in the message size due to the attachment of the policies and the monitor along with the data. This increase causes a network overhead during the transfer of the message. The overhead is reduced by piggybacking the AB with the request. The composite services can cache and store the AB in their datastore to further reduce the transfer overhead. In comparison with the traditional approaches that transmit the data and the policies but use a static monitor, this framework has an additional overhead due to the transfer of the monitor to each endpoint in the interaction. The traditional approaches also incur a network overhead because they communicate with the static monitor to evaluate and enforce the policies for data disclosure at each endpoint.

- Services have to interact with the AB to access the data which affects the response time of the service, but it is necessary to ensure policy enforcement. In comparison, the traditional approaches also have this overhead due to their interaction with either the data owner or a TTP to access the data.

- Services may experience increased resource usage due to the execution of the AB. However, the interaction time between the AB and the service is quite low so this should not have a significant impact on the performance of the service.

4. IDENTITY MANAGEMENT IN CLOUD COMPUTING

Cloud computing allows the use of Web-based services to support business processes and rental of IT services on a utility-like basis. It offers a concentration of resources, but also poses privacy risks for data in cloud. A single data breach can cause significant loss. In cloud computing, users may have multiple accounts associated with a single or multiple service providers (SPs). Entities (e.g. users, services) have to authenticate themselves to SPs in order to use their services. An entity provides personally identifiable information (PII) that uniquely identifies it to an SP. Sharing sensitive identity information along with associated attributes of the same entity across services in the cloud can lead to the mapping of the identities to the entity, tantamount to privacy loss. Identity Management (IdM) is one of the core components in cloud privacy and can help alleviate some of the problems associated with cloud computing. In this chapter, we present an entity-centric approach for IdM, which is independent of Trusted Third Party (TTP) and has the ability to use identity data on untrusted hosts. This chapter is based on the work described in "Protection of Identity Information in Cloud Computing without Trusted Third Party" [61] and "An Entity-centric Approach for Privacy and Identity Management in Cloud Computing" [62].

4.1 Introduction

The growing popularity, continuing development, and maturation of cloud computing services is an undeniable reality. Information stored locally on a computer can be stored in the cloud, including word processing documents, spreadsheets, presentations, audio, photos, videos, medical records, financial information, appointment calendars, etc. A cloud SP is a third party that maintains information about, or on

behalf of, another entity. Trusting a third party requires taking the risk of assuming that the TTP will act as it is expected (which may not be true all the time). Whenever some entity stores or processes information in the cloud, privacy or confidentiality questions may arise [63].

4.1.1 Privacy in Cloud Computing

Privacy in cloud computing can be defined as "the ability of an entity to control what information it reveals about itself to the cloud (or to the cloud-hosted SP), and the ability to control who can access that information" [61]. Numerous existing privacy laws impose the standards for the collection, maintenance, use, and disclosure of PII that must be satisfied even by cloud SPs [63]. However, due to the nature of cloud computing, there is little or no information available in a cloud to point out where data are stored, how secure they are, who has access to them, or if they are transferred to another host (if that host can be trusted). A cloud cannot be used for storing and processing data and applications if it is insecure.

An entity has to provide PII, which authenticates it while requesting services from the cloud. This leaves a trail of PII that can be used to uniquely identify, or locate a single entity, which-if not properly protected-may be exploited and abused. The major problem regarding privacy in cloud is how to secure PII from being used by unauthorized users, how to prevent attacks against privacy (such as identity theft) even when a cloud SP cannot be trusted, and how to maintain control over the disclosure of PII. Handing sensitive data to another party is a serious concern. Are data held somewhere in the cloud as secure as data protected in user-controlled computers and networks? Cloud computing can increase the risks of security breaches. Knowing who has user's personal data, how they are being accessed, and the ability to maintain control over them prevents privacy breaches of PII, and can minimize the risk of identity theft and fraud.

We achieve a solution to the privacy problem that we investigate in this chapter through the use of an entity-centric IdM approach. The approach is based on the active bundle (AB) (described in Chapter 3) – which is a middleware agent that includes PII data, privacy policies, and a monitor that enforces the policies and has a set of protection mechanisms to protect itself. An AB is used to mediate interactions between a user and cloud services. It interacts on behalf of a user to authenticate to cloud services using the privacy policies of the user. This IdM approach does not use TTPs and performs authentication without disclosing unencrypted PII. The authentication is based on computing predicates over encrypted data and multiparty computing.

4.2 Identity Management

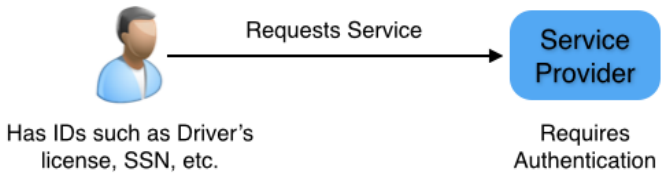

Fig. 4.1. User-SP interaction.

PII is commonly known as identity information. An identity information is a set of unique characteristics of an entity: an individual, a subject, or an object. The identity information is commonly used by entities for authentication to SPs. It provides assurance to SPs about an entity's identity, which helps the SPs to decide whether to permit the entity to use a service or not. Figure 4.1 shows an example of an interaction between a user and an SP that uses PII for authentication. In the example, the user wants to use a service; the user has to disclose some of their PII, which uniquely identifies them to the SP. However, the user does not want to disclose all of their PII. The main problem for the user is to decide which portion of the PII should be disclosed, and how to disclose it in a secure way. Since identity

information is a key for opening access to resources and paying for them, it can lead to serious crimes involving identity theft if misused or compromised [63]. An IdM system supports the management of these multiple identities. It also decides how to best disclose PII to obtain a particular service. The main tasks performed by an IdM system are as follows:

- It issues identities by associating a PII with an entity, e.g. assigning a driver's license to a person. It can also update identities, e.g. reissuing a driver's license.
- It describes identities by assigning attributes that identify an entity, e.g. date of birth (DOB) on a driver's license. It can also update the attributes associated with an identity.
- It evaluates the requests for identity verification of an entity by an SP.
- It records the usage of identities in a system and manages the identity activity logs, e.g. disclosure of a PII to an SP.
- It revokes the identities by assigning an expiration date to a PII. The use of a revoked identity fails verification.

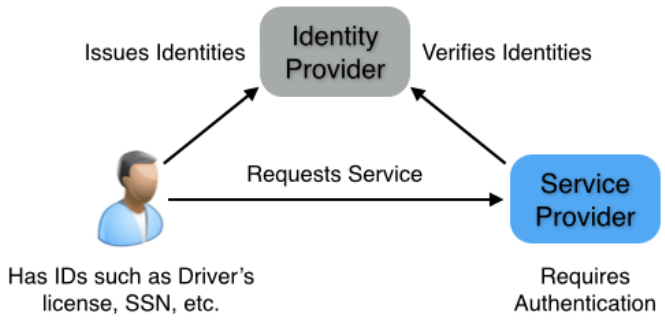

Fig. 4.2. Example of an IdM system.

An IdM system uses one of the following three categories of identifiers: (a) information that both an entity and SP know, such as passwords; (b) information that an

entity has and SP can verify it from an IdP, such as Social Security Number (SSN); and (c) information about the entity, such as fingerprints. Figure 4.2 shows a general architecture of an IdM system that uses PII for authentication. A set of parties use IdM system and collaborate[1] to identify an entity. These parties are (cf. [65]):

- An *Identity provider* (IdP) that issues digital identities, for e.g., organizations issue identities (e.g. banks issue credit cards enabling payment), governments issue identities (e.g. passports) to citizens, and online services issue identities (e.g email Ids) to the registered users.

- A *Service provider* (SP) that provides services to entities that have required identities. For instance, a user needs to provide payment and mailing information to an online service to order an item.

- An *Entity* that requests a service and has to satisfy claims by providing an identity. For instance, a claim requiring a person to be over the age of 21 can be satisfied by providing an identity with the DOB attribute.

- An *Identity verifier* (IdV) that receives requests from an SP for verifying a claim about a specific entity. It verifies the correctness and decides whether the claim is correct or not. Many times the IdV is the same party as the IdP.

4.2.1 Identity Management in Cloud Computing

The traditional model of application-centric IdM, where each application keeps track of its collection of users and manages their identities, is not acceptable in cloud-based architectures. In cloud computing, entities may have multiple accounts associated with different SPs hosted on the same cloud. Furthermore, entities may use multiple services offered by the same SP (e.g., Gmail and Google Drive are offered by Google). Figure 4.3 shows a scenario in which the IdM system and its parties (the user, SPs, IdP) are located on the same cloud. Moreover, the cloud may own some

[1]Collaboration here refers to sharing information through a predefined data exchange protocol

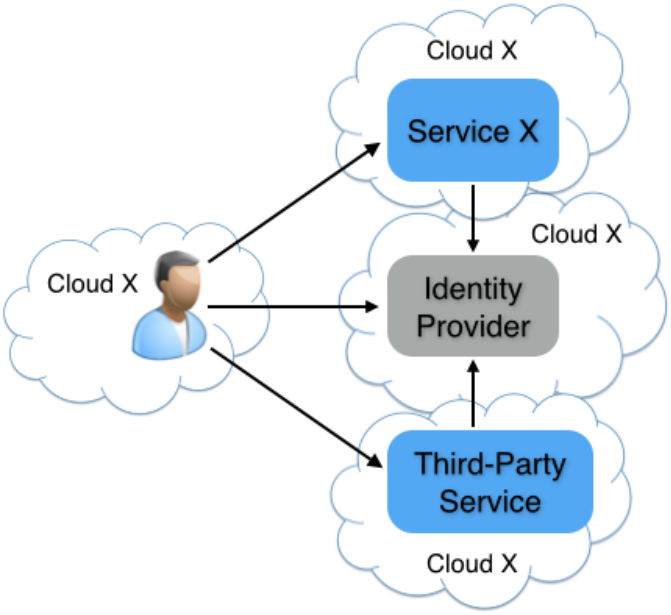

Fig. 4.3. Example of an IdM system in cloud.

or all of these services. In this case, sharing PIIs of the same entity across multiple SPs in the cloud can leave a trail of PIIs that can be used to map an entity to its PIIs and uniquely identify that entity [66].

IdM is the key to cloud privacy, but IdM in cloud is more complex than in traditional web-based systems since the users hold multiple accounts with different SPs or with a single SP. Therefore, cloud computing requires an entity-centric access control—an entity's request for a service is bundled with the entity's identity and entitlement information.

We propose an entity-centric IdM system that allows the entities: (a) to create and manage their digital identities to authenticate in a way that does not reveal their actual identities or relationships between identities to cloud providers, SPs, etc.; and (b) to protect PII from unauthorized access. The advantage of an entity-centric IdM is that the disclosure of PII is no longer arbitrary or at the will of SP but is at the will of its owner.

4.2.2 Related Work

Different solutions use different ways of sending user's PII for negotiation with the SPs. Most of the solutions that we studied use TTP for verifying or approving PII. The major issues with this approach in cloud computing are: (a) TTP could be a cloud service, so SP and TTP could be same and even located in the same cloud as the user; therefore, TTP may not be an independent trusted entity anymore; (b) using a single TTP is a centralized approach with its inherent danger that compromising TTP results in compromising all PIIs of its users as well. The existing solutions prohibit the use of untrusted hosts such that a client application holding PII must be executed on a trusted host to prevent malicious hosts from accessing PII. This may not be possible for entities using cloud hosts. A brief description of three well-known solutions for IdM follows [62].

Privacy and Identity Management for Europe (PRIME) [67] provides privacy-preserving authentication using anonymous credentials managed by an IdP. The user-side component uses protocols for getting third party (IdP) endorsements for claims to relying parties (RPs). Anonymous credentials are provided using an identity mixer protocol (based on the selective disclosure protocol) that allows users to selectively reveal any of their attributes in credentials obtained from IdP, without revealing any of their information. The credentials are digitally signed using a Public Key Infrastructure (PKI).

Windows CardSpace [68] uses tokens for digital identities. A security token consists of a set of claims, such as a username, a user's name, address, SSN, etc. The tokens prove that the claims belong to the user who is presenting them. When a CardSpace-enabled application or website wishes to authenticate a user, it requests a particular set of claims from the user. The user selects an InfoCard to use, and the Card-Space application contacts an IdP to obtain a digitally signed token that contains the requested information, which is then communicated to the requesting application.

OpenID [69, 70] is a decentralized authentication protocol that helps cloud users in managing their multiple digital identities with greater control over the sharing of their PII. A user needs to remember only one username and password–an OpenID, which can be used to logon to all websites that accept this OpenID.

4.2.3 Selected Research Problems

The research problems addressed in this work are as follows:

1. *Authenticating without disclosing PII*: When a user sends the identity information to get authenticated for a service, the user may encrypt the data. However, before this information is used by the SP, it is decrypted so that the SP can use it. But as soon as the PII is decrypted it becomes prone to attacks. This is particularly of a concern if the SP decides to store this data and is hosted on a cloud with other SPs.

2. *Using services on untrusted hosts*: The available IdM solutions need the user to be on a trusted host for using the IdM system or service. They do not recommend or allow usage of IdM systems on untrusted hosts like public hosts. With the advances in cloud computing where the users and their data may reside anywhere in the cloud (on any host), this issue needs to be addressed. For instance, the user may be on a cloud host (a Virtual Machine).

Note that the goal in this chapter is to assure that IdM does not use TTP for verifying credentials. This implies that IdM could use TTPs for other purposes, such as the use of a TTP for management of decryption keys.

4.3 Proposed IdM Approach

This section describes the proposed IdM approach, which is based on the active bundle scheme, computing predicates over encrypted data, and multiparty computing. The salient features of the approach are as follows:

- It has the ability to authenticate without disclosing unencrypted data. It uses encrypted data when negotiating the use of PII for authentication to SPs in cloud. This is achieved by evaluating predicates over encrypted data.
- It has the ability to use identity data on untrusted hosts for authentication to SPs. This is achieved through the use of the active bundle scheme.
- It is independent of TTP. This is achieved through the use of multiparty computing.

4.3.1 Predicates with Encrypted Data and Multiparty Computing

We use a predicate encryption scheme and multiparty computing for giving answers to predicates about PII. Shamir proposed threshold secret sharing [24], which works as follows. First, a secret data item D is divided into n shares $D_1, ..., D_n$. Then, a threshold k is chosen such that: (a) to recover D, k or more of arbitrary D_i shares are required; (b) using any $k-1$ or fewer D_i shares leaves D completely undetermined.

Ben-Or, Goldwasser, and Wigderson [72] define a protocol for multiparty computing of a function f using secret input from all the parties. The protocol involves n "regular" parties, which calculate only partial function outputs. In this setup [72], one of the players is selected as the dealer (denoted as DLR), and is provided the partial function outputs to find out the full results of function computation. Let f be a polynomial of degree n known to each of the n parties, and t be an arbitrary threshold value. Let P_i denote Party i, and x_i denote the secret input of P_i for f. Dealer DLR will receive from the n parties the partial outputs of f calculated by the n parties using their respective secret inputs $x_1, x_2, ..., x_n$. Let $\alpha_1, \alpha_2, ..., \alpha_n$ be distinct non-zero elements in the domain of f. The party P_i is assigned the point α_i.

Each party P_i generates a polynomial h_i of degree t (where t is the above threshold value) such that $h_i(0) = x_i$. Each P_i sends to each P_j (from the subset of the other $n-1$ parties) one share $s_{i,j} = h_i(\alpha_j)$ of P_j's input. Then, each P_i computes a portion

of function f using shares $s_{i,j}$ of the input that it has (its own) or received from the other $n-1$ parties.

A predicate encryption scheme allows evaluating predicates with encrypted data. For instance, a user Alice can compute the predicate "(email sender = 'Bob') and (date in [2014, 2015])" over encrypted email data [73].

1.	Setup	*PK*, *MSK*
2.	Encrypt(*PK*, PII)	*CT*
3.	KeyGen(*PK*, *MSK*, *p*)	TK^p
4.	Query(*PK*, *CT*, TK^p)	*p*(PII)

Fig. 4.4. Public-key predicate encryption scheme.

Figure 4.4 shows a sample predicate encryption scheme that has this property [73]. Alice uses a *Setup* algorithm to generate a public key PK and a secret key MSK. Next, Alice uses PK to encrypt (with algorithm *Encrypt*) her PII and gets ciphertext CT. Then, she can store CT (the encrypted PII) on an untrusted host (e.g., in a cloud). She may also publish PK, so that it can be used to encrypt data that she can access.

Alice has the function p representing a predicate that she wishes to evaluate for her encrypted PII. She uses the $KeyGen$ algorithm, PK, MSK, and p to output the token TK_p (encoding p). Then, she gives TK_p to the host that evaluates the token (with p included in the token) for CT (the encrypted PII), and returns the result $p(PII)$ to Alice.

Note that $KeyGen$ uses the secret key MSK as input. Hence, Alice can use $KeyGen$ to generate TK_p for p. Alice can give TK_p to an untrusted host while protecting PII. (Observe that if Alice gave $KeyGen$ and MSK to the host, the scheme would not be secure—it would not protect PII.)

For negotiating the use of a cloud service, we combine computing predicate over encrypted data with secure multiparty computing. The secret key MSK is split be-

tween n parties[2] using the above-mentioned Shamir's technique. Then, the algorithm $KeyGen$ is provided to n parties, and computed by them collaboratively using their shares of the secret key, function p representing a predicate, public key PK, and partial outputs. This is done as specified in the above-mentioned protocol of Ben-Or, Goldwasser and Wigderson for multiparty computing.

In the algorithm, a data owner O encrypts their PII using algorithm $Encrypt$ and O's public key PK. $Encrypt$ outputs CT—the encrypted PII. SP transforms its request for PII to a predicate represented by function p. Then, SP sends shares of p to the n parties that hold the shares of MSK. The n parties execute together $KeyGen$ using PK, MSK, and p, and return TK_p to SP. Next, SP calls the algorithm $Query$ that takes as input PK, CT, TK_p and produces $p(PII)$ which is the evaluation of the predicate. The owner O is allowed to use the service only if the predicate evaluates to "true". This allows a user to be authenticated without disclosing unencrypted PII.

4.3.2 Active Bundle Scheme for IdM

We briefly discuss the active bundle scheme and its use for IdM.

Overview of the Active Bundle

An active bundle (AB) is a container with a payload of sensitive data, metadata, and a monitor (EM) [38]. Sensitive data constitute content to be protected from privacy violations and unauthorized dissemination–e.g., it contains PII. Metadata describes the AB and includes the privacy policies of the data owner. The metadata can include the following components: (a) provenance metadata; (b) integrity check metadata; (c) access control metadata; (d) dissemination control metadata; (e) life duration value; (f) security metadata (including: security server id; trust server id used to validate the trust level and the role of a host; and trust level threshold required

[2]The key shares can be stored in a distributed hash table (DHT) system, such as Vuze [26], Open DHT [25]; each share can be stored in a different location.

to access data in an active bundle); and (g) other application-dependent and context-dependent metadata. The monitor (EM) manages and controls the program code enclosed in a bundle. Its main functions include (a) enforcing bundle's access control policies (e.g., disclosing to an SP only the portion of data that the SP is entitled to access); (b) enforcing bundle's dissemination policies; and (c) validating bundle integrity.

Using Active Bundle for IdM

In the proposed entity-centric model, the AB is used as a PII carrier. It provides users with control over their PII, allowing them to decide what and when PII will be shared with the SPs in the cloud. The components of an AB for IdM are as follows:

1. *Identity data* are used for authentication, getting service, and using service (e.g., SSN, DOB). These data are encrypted and packed inside the AB.

2. *Disclosure policy* is a set of rules for choosing which identity data to disclose. E.g., if an identity data I is used for service S, then I should be used each time S is accessed (minimizing disclosure of PII).

3. *Disclosure history* is used for logging and auditing purposes. It is also used for selecting identity data to be disclosed, based on previous disclosures.

4. *Monitor* contains the code/algorithm for evaluating disclosure policy. It gives access to the identity data and provides protection on untrusted hosts.

When an entity requests a service from an SP, the SP informs the entity, through a technical policy requirement, that in order to gain access to the service the user must authenticate and provide its identity information (if required). In response, an AB is created by the entity with their encrypted PII data and sent to the SP. When arriving at a "foreign" host, an AB ascertains the host's trust level through a TTP. Using its disclosure policy, it decides whether the host may be eligible to access

all or part of bundle's data, and which portion of sensitive data can be revealed to the host. An AB may realize that its security is about to be compromised. For instance, it may discover that its self-integrity check fails, or the trust level of its host is too low. In this case, the bundle does not disclose any data and can report such activity to TTP. Otherwise, the SP gets access to the encrypted PII and executes the above-mentioned algorithm to convert its verification request to a predicate token. Then, it uses the token to evaluate its predicate request over the encrypted PII. The authentication succeeds only if the predicate evaluates to "true", but does not disclose any unencrypted PII. Although one of the goals of AB is to protect sensitive data from unauthorized disclosure to malicious hosts, the scheme has its own threat model. The main element is that it requires a correct execution of an AB's EM by hosts.

4.4 Advantages of the Proposed Approach

The approach presented in this chapter is one of the alternatives to using a dedicated TTP. It reduces the risks associated with the use of TTP. The main advantages of the proposed approach are:

1. It does not need a TTP for PII management. Since data exchange between a bundle and its host is local to the host, it protects PII from man-in-the-middle, side channel, and collaborative attacks.

2. It enables authentication without disclosing unencrypted data. This prevents unnecessary data disclosures.

3. It protects identity data from untrusted hosts. If data (inside AB) reach an unintended destination or are tampered with, the disclosure is not allowed to prevent data from falling into wrong hands.

4.5 Resilience of the Proposed Approach

A system based on the proposed approach is independent of the usage of TTP. This reduces the risks of correlation attacks within a cloud. Correlation attacks on IdM happen when an entity acquires a set of data (multiple PIIs in case of IdM) and is able to correlate it to the physical identity of an entity, such as a person. In a cloud environment, the SPs and the IdP may be owned by the cloud and the entity requesting services from SPs may be present on a host in the same cloud. In this case, the approaches that use a TTP increase the risk of correlation attacks on an entity's PII. Approaches that do not use a TTP reduce the risk of such attacks.

Ristenpart et al. [74] demonstrated that the Amazon EC2 cloud is prone to side-channel attacks and it would be possible to steal data, once a malicious virtual machine is placed on the same server as its target. It is possible to carefully monitor how access to or usage of resources fluctuates and thereby potentially glean sensitive information about a victim. Though, the study points out that there are a number of factors that would make such an attack significantly more difficult in practice. Approaches that use a TTP increase the risk of side-channel attacks on an entity's PII. Approaches that do not use a TTP, such as the one that we use, reduce the risk of such attacks. However, the proposed solution is prone to other attacks, for e.g., the AB may not be executed at all by an SP. In this case, the data are not disclosed, but the entity is denied access to the service that it requests.

5. SUMMARY

There has been immense growth in the popularity of the service computing model for delivering IT solutions in critical sectors such as unmanned aerial vehicles (UAVs), online education, digital healthcare, cloud computing, product lifecycle management, etc. This model marks a shift from single-domain monolithic systems to cross-domain distributed services, which raises important privacy and security concerns for both the public and private sectors. In a cross-domain distributed system, clients interact with a primary service, which can outsource their requests (including their data) to secondary services from different ownership domains. In this case, it is very difficult for a client to determine how their data will be shared and who will access it. This invisible sharing exposes the data to new risks that are otherwise avoidable if data stays within a trusted domain. The use of third parties to handle client requests and manage data without client's knowledge or permission has significant implications for the privacy of personal, business, and governmental information.

Service providers process client requests, handle client data, and control the service they provide. They also manage policy definition (terms and conditions), data sharing, access control, and policy enforcement. Clients have to rely on service providers for the security and privacy of their data. Unauthorized data disclosures in these situations result in privacy violations which may go undiscovered. This loss of control and lack of awareness increases threats to client's data and diminishes trust in these systems. Clients could specify their requirements (policies) for data disclosure and dissemination, but current systems lack the infrastructure for transmitting those policies and ensuring their correct evaluation and enforcement in cross-domain interactions. Thus, there is a strong need for an efficient and effective solution for privacy-preserving data dissemination that can control data disclosure and protect data privacy even in external domains throughout the interaction lifecycle. The so-

lution should be able to dynamically authenticate services and authorize their data access requests by evaluating and enforcing the policies of data owners at each endpoint in a distributed interaction.

This dissertation introduced the PD3 problem and provided a formal description of the problem. It provides an overview of the existing solutions and outlines the main requirements for a new solution. The vision of the new solution responds to the notion that data are passive entities, unable to protect themselves, requiring another active and trusted entity to protect them. We challenge this assumption by proposing that data can actively protect themselves. In this vision, data can reside anywhere but are always accompanied by the owner's access policies and a policy enforcement mechanism that protects and controls their disclosure. The solution is realized through the EPICS framework—an innovative solution for cross-domain data dissemination and policy enforcement in composite Web services. The dissertation describes the design and implementation of the framework and discusses the performance and security evaluation using a realistic e-commerce scenario. It presents as an application of the framework a novel approach for privacy and identity management in cloud computing. The proposed framework is compatible with existing service infrastructure and meets the real-time constraints of Web service interactions.

In the future, this work can be extended in several ways: (a) designing new policy languages and policy evaluation engines, (b) developing mechanisms for trust management of services, (c) extending the framework implementation with stateful and mutable versions of active bundles, (d) using data access, disclosure, and dissemination logs of active bundles for anomaly detection and trust evaluation, (e) enhancing security of the framework against execution hijack attacks using mechanisms such as code obfuscation, (f) developing dynamic data-filtering mechanisms to support fine-grained data disclosure, (g) applying the framework for data dissemination and policy enforcement in other domains such as vehicle-to-vehicle networks, digital rights management, smart grid systems, etc.

REFERENCES

REFERENCES

[1] "Target data breach," https://corporate.target.com/about/shopping-experience/payment-card-issue-FAQ, accessed: June 2015.

[2] "Anthem data breach," http://www.anthemfacts.com, accessed: June 2015.

[3] "Sony pictures entertainment hack," http://en.wikipedia.org/wiki/Sony_Pictures_Entertainment_hack, accessed: June 2015.

[4] R. Ranchal, A. Mohindra, J. Manweiler, and B. Bhargava, "Radical strategies (RADS) for engineering web-scale cloud solutions," *IEEE Cloud Computing*, 2015, to appear.

[5] R. Ranchal, A. Mohindra, N. Zhou, S. Kapoor, and B. Bhargava, "Hierarchical aggregation of consumer ratings for service ecosystem," in *Proceedings of the 22nd IEEE International Conference on Web Services*, 2015, to appear.

[6] R. Ranchal and B. Bhargava, "Protecting PLM data throughout their lifecycle," in *Proceedings of the 9th Quality, Reliability, Security and Robustness in Heterogeneous Networks (QSHINE)*, 2013, pp. 633–642.

[7] B. Bhargava, R. Ranchal, and L. Ben Othmane, "Secure information sharing in digital supply chains," in *Proceedings of the 3rd IEEE International Advance Computing Conference*, 2013, pp. 1636–1640.

[8] R. Khalaf, M. Kim, A. Mohindra, V. Muthusamy, R. Ranchal, V. Salapura, and A. Slominski, "Building scalable, secure, multi-tenant cloud services on IBM Bluemix," *IBM Journal of Research & Development*, 2015, to appear.

[9] W. A. Yasnoff, L. Sweeney, and E. H. Shortliffe, "Putting health IT on the path to success," *The Journal of the American Medical Association*, vol. 309, no. 10, pp. 989–990, 2013.

[10] J. Viega and J. Epstein, "Why applying standards to Web services is not enough," *IEEE Security & Privacy*, vol. 4, no. 4, pp. 25–31, 2006.

[11] M. Azarmi, B. K. Bhargava, P. Angin, R. Ranchal, and N. Ahmed, "An end-to-end security auditing approach for service oriented architectures." in *Proceedings of the 31st IEEE Symposium on Reliable Distributed Systems*, 2012, pp. 279–284.

[12] Y. G. Le Gall, A. J. Lee, and A. Kapadia, "PlexC: A policy language for exposure control," in *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, 2012, pp. 219–228.

[13] R. Ranchal, D. Ulybyshev, P. Angin, and B. Bhargava, "PD3: Policy-based distributed data dissemination," *16th CERIAS Information Security Symposium*, 2015, poster.

[14] H. Abelson, R. Anderson, S. M. Bellovin, J. Benalob, M. Blaze, W. Diffie, J. Gilmore, P. G. Neumann, R. L. Rivest, J. I. Schiller, and B. Schneier, "The risks of key recovery, key escrow, and trusted third-party encryption," *World Wide Web Journal*, vol. 2, no. 3, pp. 241–257, 1997.

[15] J. Katz and Y. Lindell, "Public-key (asymmetric) cryptography," in *Introduction to Modern Cryptography*. CRC Press, 2014.

[16] G. Denker, J. Millen, and Y. Miyake, "Cross-domain access control via PKI," in *Proceedings of the 3rd IEEE International Workshop on Policies for Distributed Systems and Networks*, 2002, pp. 202–205.

[17] D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing," in *Advances in Cryptology*, 2001, pp. 213–229.

[18] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006, pp. 89–98.

[19] F. B. Schneider, "Enforceable security policies," *ACM Transactions of Information and System Security*, vol. 3, no. 1, pp. 30–50, 2000.

[20] R. E. Smith, *Authentication: From Passwords to Public Keys*. Addison-Wesley Longman Publishing, 2001.

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[22] L. Wang, D. Wijesekera, and S. Jajodia, "A logic-based framework for attribute based access control," in *Proceedings of the ACM Workshop on Formal Methods in Security Engineering*, 2004, pp. 45–55.

[23] R. Geambasu, T. Kohno, A. A. Levy, and H. M. Levy, "Vanish: Increasing data privacy with self-destructing data." in *Proceedings of the USENIX Security Symposium*, 2009, pp. 299–316.

[24] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[25] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: A public DHT service and its uses," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2005.

[26] "Azureus Vuze," http://www.vuze.com/, accessed: June 2015.

[27] P. Angin, "Autonomous Agents-Based Mobile-Cloud Computing," Ph.D. dissertation, Purdue University, West Lafayette, IN, USA, 2013.

[28] H. Chang and M. J. Atallah, "Protecting software code by guards," in *Security and Privacy in Digital Rights Management*, 2002, pp. 160–175.

[29] R. Ranchal, B. Bhargava, L. Othmane, and P. Angin, "EPICS: A framework for enforcing policies in composite web services," *IEEE Transactions on Parallel and Distributed Systems*, 2015, submitted.

[30] A. Cockcroft, "Netflix cloud architecture," http://www.qcontokyo.com/pdf/adriancockroft.pdf, accessed: June 2015.

[31] "Amazon architecture." http://highscalability.com/amazon-architecture, accessed: June 2015.

[32] D. Namiot and M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–27, 2014.

[33] R. M. Salih, L. Lilien, and L. B. Othmane, "Protecting patients' electronic health records using enhanced active bundles," in *Proceedings of the 6th International Conference on Pervasive Computing Technologies for Healthcare, Doctoral Consortium*, 2012, pp. 1–4.

[34] M. P. Papazoglou and W.-J. Van Den Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The International Journal on Very Large Data Bases*, vol. 16, no. 3, pp. 389–415, 2007.

[35] V. Suhendra, "A survey on access control deployment," in *Security Technology*. Springer Berlin Heidelberg, 2011, pp. 11–20.

[36] D. Basin, V. Jugé, F. Klaedtke, and E. Zălinescu, "Enforceable security policies revisited," *ACM Transactions on Information and System Security*, vol. 16, no. 1, pp. 3:1–3:26, 2013.

[37] "eXtensible access control markup language (XACML) version 3.0," http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, accessed: June 2015.

[38] L. Ben Othmane and L. Lilien, "Protecting privacy of sensitive data dissemination using active bundles," in *Proceedings of the IEEE World Congress on Privacy, Security, Trust and the Management of e-Business*, 2009, pp. 202–213.

[39] L. Ben Othmane, "Active Bundles for Protecting Confidentiality of Sensitive Data Throughout Their Lifecycle," Ph.D. dissertation, Western Michigan University, Kalamazoo, MI, USA, 2010.

[40] B. Bhargava, P. Angin, R. Ranchal, R. Sivakumar, M. Linderman, and A. Sinclair, "A trust based approach for secure data dissemination in a mobile peer-to-peer network of AVs," *International Journal of Next-generation Computing*, vol. 3, no. 1, 2012.

[41] C. Collberg and J. Nagra, *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley Professional, 2009.

[42] "Security assertion markup language (SAML)," https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, accessed: June 2015.

[43] S. Cantor and T. Scavo, "Shibboleth architecture," *Protocols and Profiles*, vol. 10, p. 16, 2005.

[44] J. Park, R. Sandhu, and J. Schifalacqua, "Security architectures for controlled digital information dissemination," in *Proceedings of the 16th IEEE Annual Computer Security Applications Conference*, 2000, pp. 224–233.

[45] O. Sibert, D. Bernstein, and D. Van Wie, "Digibox: A self-protecting container for information commerce," in *Proceedings of the USENIX Workshop on Electronic Commerce*, 1995, pp. 1–13.

[46] Y.-Y. Chen, P. A. Jamkhedkar, and R. B. Lee, "A software-hardware architecture for self-protecting data," in *Proceedings of the ACM Conference on Computer and Communications Security*, 2012, pp. 14–27.

[47] F. Li, B. Luo, P. Liu, D. Lee, and C.-H. Chu, "Enforcing secure and privacy-preserving information brokering in distributed information sharing," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 6, pp. 888–900, 2013.

[48] S. Pearson and M. C. Mont, "Sticky policies: An approach for managing privacy across multiple parties," *IEEE Computer*, no. 9, pp. 60–68, 2011.

[49] M. C. Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services," in *Proceedings of the 14th IEEE International Workshop on Database and Expert Systems Applications*, 2003, pp. 377–382.

[50] "Cloud computing and security - a natural match," http://www.trustedcomputinggroup.org, Trusted Computing Group, Tech. Rep., 2010.

[51] B. J. Parno, "Trust Extension as a Mechanism for Secure Code Execution on Commodity Computers," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2010.

[52] M. C. Mont, K. Harrison, and M. Sadler, "The HP time vault service: Exploiting IBE for timed release of confidential information," in *Proceedings of the 12th ACM International conference on World Wide Web*, 2003, pp. 160–169.

[53] L. Lilien and B. Bhargava, "A scheme for privacy-preserving data dissemination," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 36, no. 3, pp. 503–506, 2006.

[54] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. Shah, "First experiences using XACML for access control in distributed systems," in *Proceedings of the ACM Workshop on XML Security*, 2003, pp. 25–37.

[55] H. Krawczyk, "Cryptographic extraction and key derivation: The HKDF scheme," in *Advances in Cryptology*, 2010, pp. 631–648.

[56] "Web services policy 1.5 - framework," http://www.w3.org/TR/ws-policy/, accessed: June 2015.

[57] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[58] "IBM Bluemix," https://console.ng.bluemix.net, accessed: June 2015.

[59] "Google App Engine," https://cloud.google.com/appengine/docs, accessed: June 2015.

[60] J. Torrey, "HARES: Hardened anti-reverse engineering system," in *Proceedings of the Symposium on Security for Asia Network*, 2015.

[61] R. Ranchal, B. Bhargava, L. B. Othmane, L. Lilien, A. Kim, M. Kang, and M. Linderman, "Protection of identity information in cloud computing without trusted third party," in *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*, 2010, pp. 368–372.

[62] P. Angin, B. Bhargava, R. Ranchal, N. Singh, M. Linderman, L. B. Othmane, and L. Lilien, "An entity-centric approach for privacy and identity management in cloud computing," in *Proceedings of the 29th IEEE Symposium on Reliable Distributed Systems*, 2010, pp. 177–183.

[63] R. Gellman, "Privacy in the clouds: risks to privacy and confidentiality from cloud computing." in *Proceedings of the World Privacy Forum*, 2012.

[64] A. Jøsang and S. Pope, "User centric identity management," in *Proceedings of the 4th Asia Pacific Information Technology Security Conference*, 2005, pp. 77–89.

[65] K. Cameron and M. B. Jones, "Design rationale behind the identity metasystem architecture," in *Proceedings of the ISSE/SECURE Securing Electronic Business Processes*, 2007, pp. 117–129.

[66] A. Gopalakrishnan, "Cloud computing identity management," *SETLabs briefings*, vol. 7, no. 7, pp. 45–55, 2009.

[67] J. Camenisch, D. Sommer, S. Fischer-Hübner, M. Hansen, H. Krasemann, G. Lacoste, R. Leenes, J. Tseng *et al.*, "Privacy and identity management for everyone," in *Proceedings of the ACM Workshop on Digital Identity Management*, 2005, pp. 20–27.

[68] W. A. Alrodhan and C. J. Mitchell, "Improving the security of cardspace," *EURASIP Journal on Information Security*, vol. 9, 2009.

[69] "OpenID," http://openid.net/, accessed: June 2015.

[70] D. Recordon and D. Reed, "OpenID 2.0: a platform for user-centric identity management," in *Proceedings of the 2nd ACM workshop on Digital Identity Management*, 2006, pp. 11–16.

[71] B. Laurie, "OpenID: Phishing heaven," http://www.links.org/?p=187, accessed: June 2015.

[72] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988, pp. 1–10.

[73] E. Shi, "Evaluating Predicates Over Encrypted Data," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2008.

[74] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications security*, 2009, pp. 199–212.

VITA

VITA

Rohit Ranchal received his Ph.D. degree in computer science from Purdue University in August 2015 under the direction of Professor Bharat Bhargava. His research interests lie in the areas of cloud and service computing, security and privacy, and distributed systems. He received his M.S. degree in computer science from Purdue University in 2011 and B.Tech. degree in information technology from Punjab Technical University, India in 2009. He worked as an intern at the IBM India Research Lab in 2013, and at the IBM T.J. Watson Research Center in 2014. He received the ACM Graduate Teaching Assistant Award in 2013 and Raymond Boyce Graduate Teacher Award and Teaching Academy Graduate Teaching Award in 2014. He was selected for a Graduate Teaching Fellowship in 2014. His research poster won first prize (out of 43 posters) at the 16th CERIAS Information Security Symposium in 2015.