Secure Data Exchange and Data Leakage Detection in an Untrusted Cloud

Denis Ulybyshev(✉), Bharat Bhargava, and Aala Oqab-Alsalem

Computer Science Department, CERIAS, Purdue University,
West Lafayette 47907, USA
{dulybysh, bbshail, alsalema}@purdue.edu

Abstract. In service-oriented architecture, services can communicate and share data amongst themselves. It is necessary to provide role-based access control for data. In addition, data leakages made by authorized insiders to unauthorized services should be detected and reported back to the data owner. In this paper, we propose a solution that uses role- and attribute-based access control for data exchange among services, including services hosted by untrusted environments. Our approach provides data leakage prevention and detection for multiple leakage scenarios. We also propose a damage assessment model for data leakages. The implemented prototype supports a privacy-preserving exchange of Electronic Health Records that can be hosted by untrusted cloud providers, as well as detecting leakages made by insiders.

Keywords: Data leakage detection · Access control · Privacy
Cloud security

1 Introduction

Services in Service-Oriented Architecture (SOA) can communicate and share data amongst themselves. A methodology for privacy-preserving data exchange among services in SOA, in which each service can access only those data items the service is authorized for, was presented in [3, 14]. In this paper, we extend that approach with data leakage detection capabilities and a damage assessment model for data leakages. Each service can have a database associated with it. Our solution handles non-relational databases stored as key-value pairs. The methodology employs Active Bundles (AB) [2, 3, 14], that contain key-value pairs with values in encrypted form; metadata; access control policies and a policy enforcement engine (virtual machine) [3]. Each subset of data (e.g. contact, medical or billing information of a patient) is encrypted with its own symmetric key, using an AES encryption scheme. Our solution supports both centralized and decentralized data exchanges. The Active Bundle can be hosted by the server or cloud provider that serve data requests. We also support fully decentralized architecture, when services in a peer-to-peer network exchange data by sending Active Bundle amongst themselves. Default implementation of Active Bundles used in the 'WAXEDPRUNE' project [1, 5] provides privacy-preserving data dissemination among services, but does not protect against data leakages made by insiders to unauthorized entities. In this paper, we address two scenarios of data leakages: leakage of the whole Active Bundle and leakage of the plaintext data to unauthorized services. In the first scenario, leakage is prevented by an Active Bundle kernel, which prohibits unauthorized data accesses and, in addition, contains digital watermark. The second scenario is more challenging since protection provided by the Active Bundle is removed in this case. As a solution, we employ digital and visual watermarks, as well as monitoring network traffic between web services in SOA. A web crawler with built-in classifier detects digital watermark, embedded into RGB images, if the leaked image is located in public network directory. Digital watermarks for RGB images are based on a pixel transformation function. Monitoring network messages and validating data packets with a specific pattern allows detecting data leakage for cases when data pattern can be validated. For instance, credit card number always follows the specific pattern that can be validated using regular expressions [18]. We embed visual watermarks on a web page when data retrieved from an Active Bundle are displayed in a client's browser. Zoomed visual watermarks can be used as evidence of data ownership. Our approach helps to investigate data leakages and do forensics based on provenance records that are made each time a data request is served by the Active Bundle. Provenance [8] records contain information regarding who is trying to access what class of data, when, and the origin of the Active Bundle.

The rest of the paper is organized as follows: Sect. 2 contains a brief overview of related work. Section 3 describes the core design. Section 4 presents the evaluation. Section 5 concludes the paper.

2 Related Work

There are variety of Digital Rights Management (DRM) protection tools [11] that provide role-based access control. These tools are supposed to protect against data leakages. Microsoft has DRM-service, called Windows Media DRM [12]. It is designed to provide dissemination of audio and video content over an IP network. The "MediaSnap©" DRM solution [24] was proposed to protect PDF documents. Most of its principles are applicable to other digital media content. The core component of the "MediaSnap©" system is a pdf-plugin. Our data exchange model considers the context and client's attributes, such as the trust level, which is constantly recalculated, cryptographic capabilities of a browser and authentication method. A Digital Cosine Transformation [9] can be used to create watermarks for images.

The hardware-based DRM approach provides a trusted hardware space for executing only permitted applications. "DRM services such as content decryption, authentication and rights rendering take place only in this trusted space" [11]. Advantages of hardware-based DRM are that it is resistant to security breaches in used operating systems, it is infeasible to bypass security features and it provides memory space protection. The main disadvantages are higher costs, limited flexibility and less interoperability [13].

Ranchal et al. [10] proposed a Framework for Enforcing Security Policies in Composite Web Services (EPICS), which protects data privacy throughout the service interaction lifecycle. The framework uses the Active Bundle concept [3, 14] for SOA. The solution ensures that the data are distributed along with the access control policies and with an execution monitor that controls data disclosures. The framework provides cross-domain privacy-preserving data dissemination in untrusted environments and reduces the risk of unauthorized access. We extend that approach with capabilities of detecting data leakages made by authorized insiders to unauthorized entities. We also consider wider set of attributes in attribute-based access control model.

Nevase et al. [19] proposed a steganography-based approach to detect leakages of images, text, video and audio content. Steganography provides covert communication channel between data entity and data owner by hiding the message in the sensitive content, e.g. in the image or text. The existence of sensitive message in the content is hidden for everyone except the data owner, who is able to decipher it. Steganography-based "forensic readiness model" [20] identifies and prevents emails, which attempt to leak data. Kaur et al. [21] also addressed prevention of data leakages that can be made by malicious insiders via emails. Email is protected via gateway during data transfer. The algorithm matches the email pattern with the stored keywords in order to detect leakage and take the action to prevent it.

Gupta and Singh [22] presented an approach for detecting intentional and inadvertent data leakages using a probabilistic model. Detecting a data leaking malicious entity is based on the allocation and distribution of data items among the agents using Bigraph.

3 Core Design

3.1 Data Leakage Detection

In our solution web services exchange data by means of *Active Bundles* [2, 3, 14]. Active Bundle is the core component that provides data leakage prevention in our system. Active Bundle is a self-protected structure that includes key-value pairs with encrypted values, access control policies, metadata and policy enforcement engine (virtual machine) [2, 3, 14]. Each subset of data (e.g. contact information of a patient, billing or medical information such as medical history, test results, diagnosis, prescriptions, etc.) is encrypted with its own symmetric key, using an AES encryption scheme. The key is generated on-the-fly using unique information produced in the execution control flow of an Active Bundle [3], depending on the subject's (service's) role, extracted from X.509 certificate of the subject (e.g. doctor, insurance agent or researcher), set of access control policies and on Active Bundle code, including authentication and authorization code. Details of the on-the-fly key derivation procedure are covered in [3]. One of the novelties offered by Active Bundle concept is that the symmetric keys used to encrypt/decrypt sensitive data are not stored neither on a cloud provider nor inside an Active Bundle nor on any Trusted Third Party (TTP). Firstly, the identity of service requesting data from an Active Bundle is verified. Services present their X.509 certificates signed by a trusted Certificate Authority (CA) to the Active Bundle [3]. Then, if authentication is granted, the client's attributes, such as browser's cryptographic capabilities, are evaluated. Then, access control policies stored in the Active Bundle are evaluated. Based on the evaluations made by the Active Bundle kernel, symmetric decryption keys are created to decrypt the accessible data items. Access control policies are enforced by the open-source policy enforcement engine “Balana” [4]. Based on derived decryption keys, the values from corresponding key-value pairs belonging to accessible data subsets are decrypted and sent back to the client by means of an https protocol. If a doctor requests for medical data, contact and billing information, the data will be extracted from Active Bundle, decrypted and sent back to the doctor. However, if the doctor sends a data request from an outdated and insecure browser, then a lesser portion of the medical data will be retrieved from the Electronic Health Record (EHR) of a patient. Each EHR is stored as an Active Bundle, one per patient. One of the key-value pairs (with encrypted value) from our Active Bundle is given below:

```
{ ″ab.patientID″ : ″Enc(001122)″ }
```

A Javascript object notation (JSON) [7] is used to store key-value pairs, where value is encrypted with a symmetric key generated for each data class. An Active Bundle has a built-in tamper-resistance mechanism, which is based on the digest of the Active Bundle components and their resources. This digest is calculated when an Active Bundle is created. Modification of any of the components’ resources will lead to a different digest and to incorrect decryption key generation. It protects from an attacker that tries to:

(a) modify an Active Bundle code in order to bypass authentication phase or evaluation of access control policies and a client’s attributes;
(b) modify access control policies in order to get access to unauthorized data;
(c) impersonate service identity by using the wrong certificate in order to get access to unauthorized data.

Instead of data, Active Bundles can store software modules in encrypted form. For instance, different departments within an organization may have different permission levels for software access and updates. Our approach guarantees that each software module can only be accessed by authorized entities.

Our methodology for data exchange in SOA supports both centralized and decentralized architectures. An Active Bundle can be hosted by a server or cloud provider that serves data requests for the Active Bundle. We also support fully decentralized architecture, when services in peer-to-peer network can exchange data by sending Active Bundle amongst each other. That is why all data that can be shared among services are included in encrypted form into an Active Bundle, but unauthorized data requests are denied, based on client’s role and attributes, such as the cryptographic capabilities of the browser [15, 16] and trust level. To demonstrate the core design of our approach, we consider a hybrid architecture when both centralized and decentralized data exchanges among services are supported (see Fig. 1). A demo video of the implemented prototype for an EHR management system is available [17]. There is a Hospital Information System (IS), hosted by the cloud provider. It hosts EHRs of patients as Active Bundles, one Active Bundle per patient. The EHR of a patient consists of 3 types of data: contact, medical and billing information. There are three services (subjects): Doctor, Insurance Agent and Researcher, who can send data requests to the Hospital Information System, i.e. to Active Bundles (see Fig. 1). The access control matrix is given in Table 1.

Adversary Model:

1. Cloud provider or server may have curious or malicious administrator that tries to access confidential data or modify them.
2. Client can be malicious in terms of:
   a. leaking data, for which client is authorized, to unauthorized parties;
   b. modifying the Active Bundle code to extract the confidential data for which the client is unauthorized.

Table 1. Access control matrix for EHR

Role/Data class	Medical data	Contact information	Billing information
Doctor	Allow	Allow	Allow
Insurance agent	Deny	Allow	Allow
Researcher	Allow	Deny	Allow

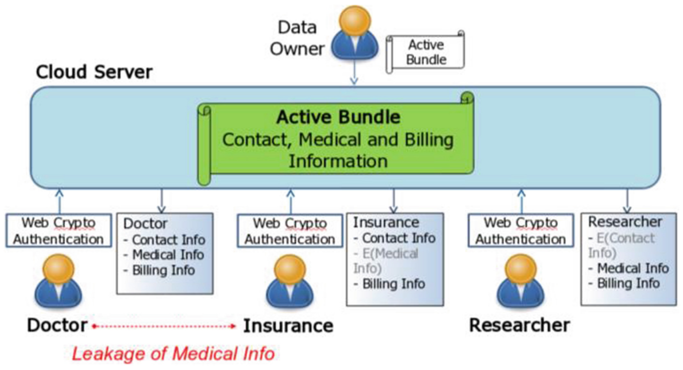

Data Owner
Active Bundle
Cloud Server
Active Bundle
Contact, Medical and Billing Information
Web Crypto Authentication
Doctor - Contact Info - Medical Info - Billing Info
Web Crypto Authentication
Insurance - Contact Info - E(Medical Info) - Billing Info
Web Crypto Authentication
Researcher - E(Contact Info) - Medical Info - Billing Info
Doctor
Insurance
Researcher
Leakage of Medical Info

Fig. 1. Hospital information system (proposed by Dr. Leon Li, NGC)

Assumptions:

1. The entity that hosts/executes Active Bundle has trusted hardware, a trusted operating system and a trusted Java Virtual Machine.
2. Http(s) protocol is used for data exchanges amongst all the web services.
3. Leaked data is attempted to be used by the adversary that has it, i.e. the adversary tries to decrypt data from a leaked Active Bundle or uploads the leaked Active Bundle to a publicly available network directory. The analogy is that someone, who is using an unlicensed copy of the Microsoft Windows XP operating system, tries to update it from an official Microsoft repository.

4. Leaked data is accessible on the adversary's side. In case of investigating leakage incident by using visual watermarks, the investigator needs permission to search for leaked data, e.g. having a police order to examine a suspicious hard drive.

Default implementation of Active Bundles used in [1], does not protect against malicious insiders that leak data to unauthorized entities. We address two leakage scenarios in our extended solution below: leakage of the whole EHR in the form of Active Bundle and leakage of decrypted plaintext data without the Active Bundle. The following implemented features provide leakage detection/prevention:

(a) Active Bundle that stores data in encrypted form and access control policies.
(b) Digital watermark, embedded into Active Bundle.
(c) Digital watermark, embedded into RGB images, stored in Active Bundle.
(d) Visible and nearly invisible visual watermarks, used to display data.
(e) Monitoring network messages and validating data packets with a specific pattern, e.g. credit card number pattern, using regular expressions [18].

The limitation of digital/visual watermarking approach, used in methods (b), (c) and (d), is that it does not work once watermark is removed from the data. Ways to mitigate and prevent data leakages in this case are proposed in Sect. 3.1.3 below.

3.1.1 Leakage of the Active Bundle

In our scenario, illustrated in Fig. 1, clients are allowed to store Active Bundles locally. For instance, a doctor might want to store the EHR of a patient on her local department computer in the hospital for cases in which the hospital Information System is down. When an Active Bundle is saved locally, the identity (or role) of the subject who saved it is written into the Active Bundle in encrypted form. This identity will be used to detect data leakage ("Sender Role" column in Table 2). The feature to save the EHR locally can be disabled, if necessary. In that case, this type of potential data leakage will be prevented, but the network architecture becomes centralized with having central storage of EHRs as a single point of failure. After saving the EHR on her local repository, the doctor can inadvertently or intentionally send the EHR to a service that is not authorized for some included data items, e.g. to an insurance service. If the insurance agency tries to access detailed medical data (e.g. X-Ray or blood test results), this access will be denied by the Active Bundle, since access control policies embedded into the Active Bundle don't permit insurers to read the medical data of a patient; they only allow them to read contact and billing information. An attempt to decrypt data made by an unauthorized service will be recorded by a trusted Central Monitor (CM). Every Active Bundle transaction in the data exchange network is monitored by the CM, who is notified each time a client tries to decrypt a data subset from an Active Bundle. The notification message contains information on what service attempts to decrypt what type of data, when, who is the origin/sender of the Active Bundle. The Central Monitor queries its local database of access control policies, called *data obligations,* in order to check whether the service that tries to decrypt data is authorized for that class of data. Without obtaining permission from the trusted Central Monitor, the data decryption process will not continue. Figure 2 illustrates the process. An Active Bundle contains encrypted data **Enc [Data(D)] = {Enc$_{k1}$ (d$_1$), …, Enc$_{kn}$ (d$_n$)}** and access control policies **(P) = {p$_1$, … , p$_k$}**. Service M is authorized to read d_i and it may leak decrypted d_i (addressed in the Sect. 3.1.2) or the entire Active Bundle to service N, who is unauthorized to access d_i. If N attempts to decrypt d_i, the Active Bundle kernel sends a message to CM to verify whether d_i is supposed to be at N. In addition, service N might be asked to get an activation code from the authentication server, which is under our control, and which will again notify the Central Monitor that data of type d_i has arrived from service M to service N. If d_i is not supposed to be at N then:

(a) trust level of services M and N is decreased;
(b) data d_i is marked as compromised and all other services are notified about that;
(c) Active Bundle is re-created with stricter access control policies to make it stronger against similar leakages:
  (c1) separate compromised role (of service M) into *Role* and *Trustworthy_Role*;
  (c2) send new certificates with *Trustworthy_Role* to all trustworthy entities;
  (c3) create a new Active Bundle with modified policies to prohibit data access for *Role*;
  (c4) disable the “Save As” functionality to prohibit storing sensitive data locally;
  (c5) raise the sensitivity level for leaked data types to prevent leakage repetition.

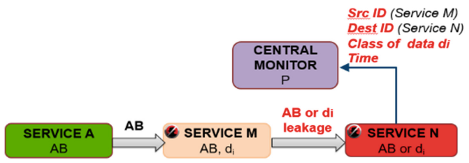


Fig. 2. Data leakage detection by central monitor

Table 2. Data obligations (access control policies)

Recipient role	Sender role	Data type	Access result
Doctor	Doctor	All	Allow
Doctor	Insurance	All	Allow
Doctor	Researcher	All	Allow
Insurance	Doctor	All	Deny
Insurance	Researcher	All	Deny
Researcher	Doctor	All	Deny
…	…	…	…
Insurance	Doctor	Medical	Deny
Insurance	Doctor	Contact	Allow
Researcher	Doctor	Contact	Deny
…	…	…	…
Researcher	Insurance	Contact	Deny
Doctor	Researcher	Contact	Deny
Insurance	Researcher	Contact	Deny

Service trust level is calculated by CM, based on the following parameters: (a) number of sent/received data requests, (b) number of rejected data requests, (c) number of communication errors, (d) CPU/Memory usage.

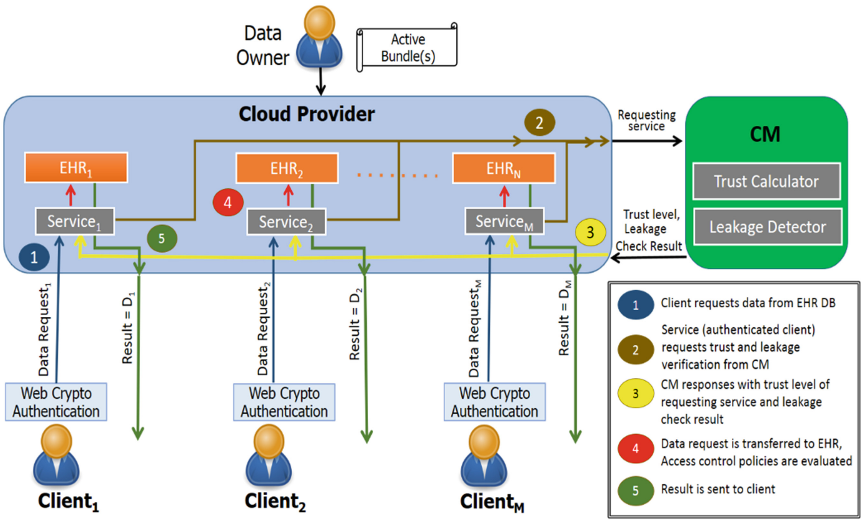

Data Owner
Active Bundle(s)
Cloud Provider
EHR$_1$
EHR$_2$
EHR$_N$
Service$_1$
Service$_2$
Service$_M$
Requesting service
CM
Trust Calculator
Leakage Detector
Trust level, Leakage Check Result
Data Request$_1$
Result = D$_1$
Data Request$_2$
Result = D$_2$
Data Request$_M$
Result = D$_M$
Web Crypto Authentication
Client$_1$
Client$_2$
Client$_M$
1 Client requests data from EHR DB
2 Service (authenticated client) requests trust and leakage verification from CM
3 CM responses with trust level of requesting service and leakage check result
4 Data request is transferred to EHR, Access control policies are evaluated
5 Result is sent to client

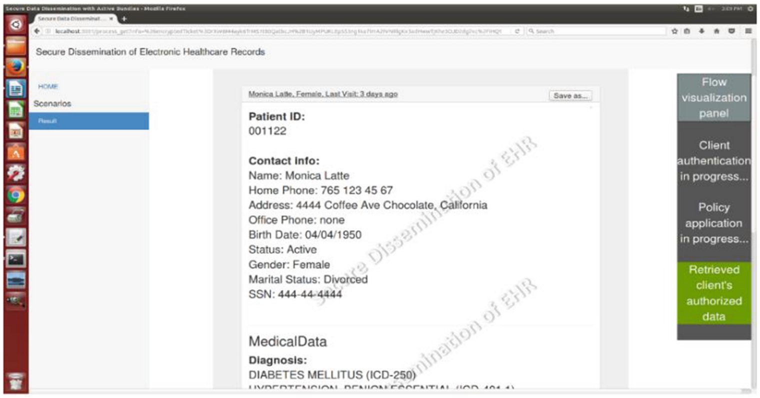
Fig. 3. Secure EHR sharing framework with data leakage detection capability

If the trust level goes below the specified threshold, future data requests coming from that service to the Active Bundle will be denied, even if access control policies allow that service to access a certain data item. Figure 3 illustrates the control flow for data request with added data leakage detection capability. As shown above, a data request is processed by the Leakage Detector and Trust Calculator. If a leakage is detected or the trust level is not sufficient, the data request is denied; otherwise, the data request will be transferred from the Central Monitor service to the Active Bundle (EHR), where a client's attributes and access control policies are evaluated. The Central Monitor hosts the relational database of obligations, i.e. of access control policies. To make a data leakage check, the Central Monitor issues a SQL query to this database. An example of the database used in our scenario with EHR sharing is given in Table 2.

In addition to the data obligations enforced by trusted Central Monitor, we implemented a web crawler to verify digital watermarks that are embedded into the Active Bundles. If an Active Bundle is uploaded to a publicly available directory in the network, the crawler verifies the digital watermark to check whether Active Bundle is supposed to be at that network node. We assume that it is possible to determine the identity of the node that hosts a public directory (e.g. in the Hospital Intranet). Network nodes, participating in data exchanges, use X.509 certificates that identify their roles (e.g. doctor or insurance agent or researcher). Also, identity can be based on node's IP address or other attributes.

3.1.2 Leakage of Plaintext Data

A second, more challenging data leakage scenario that we address in this work, is when the service that is authorized for data di can get it from an Active Bundle (see Fig. 2), store it locally in plaintext form and then send it behind the scene as a plaintext to an unauthorized service without the Active Bundle. Even if local storage functionality is prohibited, an authorized client (malicious insider) can still take a picture of a displayed di on a mobile phone's camera when di is displayed on the screen.


Secure Data Dissemination with Active Bundles - Mozilla Firefox
Secure Data Disseminat...
localhost
Search
HOME
Scenarios
Result
Save as...
Flow visualization panel
Client authentication in progress...
Policy application in progress...
Retrieved client's authorized data
Secure Dissemination of EHR
Secure Dissemination of EHR

Secure Dissemination of Electronic Healthcare Records

Monica Latte, Female, Last Visit: 3 days ago

Patient ID:
001122

Contact Info:
Name: Monica Latte
Home Phone: 765 123 45 67
Address: 4444 Coffee Ave Chocolate, California
Office Phone: none
Birth Date: 04/04/1950
Status: Active
Gender: Female
Marital Status: Divorced
SSN: 444-44-4444

MedicalData

Diagnosis:
DIABETES MELLITUS (ICD-250)

Fig. 4. Retrieved medical information on a web page with visual watermarks

Protection provided by Active Bundle is gone in this case, and we cannot prevent plaintext di leakage. We aim to help investigating the leakage and do forensics based on provenance records that are stored on the trusted Central Monitor each time a data request is served by the Active Bundle. Provenance [8] records contain information on who is trying to access what class of data, when, and who is the origin/sender of that Active Bundle. To mitigate a leakage problem, we embed visual watermarks on a web page when data retrieved by client from an Active Bundle is displayed in the client's browser. Zoomed visual watermarks can be used as an evidence of data ownership. There are two types of visual watermarks that we use to display data: clearly visible (see text "Secure Dissemination of EHR" on Fig. 4) and very small ones that are only visible if zoomed. These watermarks will remain on the image if picture of a screen is taken by a malicious client. For some types of data, e.g. on Fig. 4, it is easy to reproduce the screen's content and write it down e.g. on a piece of paper. It removes the visual watermark. But for some types of medical data, e.g. X-Ray images, it is hard to reproduce them on a piece of paper and easier to take picture of a screen, which will contain both our visual watermarks: large visible and small invisible.

Additionally, we embed digital watermarks into RGB images. The conversion function F (r, g, b) is applied to every pixel. It changes the RGB image in such a way that it is indistinguishable by the human eye from the original RGB image. If we add +1 to the RGB values of every pixel, if the value is less than 255, it will not be distinguishable to human eye. However, our web crawler, which has a built-in classifier, is able to determine whether the RGB image has the embedded watermark or not. We assume an RGB values range from 0 to 255. This watermarking method works only if the RGB image is stored in a publicly available folder. The simple way to modify an RGB image is to change the RGB values for every pixel by adding or subtracting 1 in such a way that the sum of RGB values is always odd for every pixel. Initial values should be less than 255 for adding and greater than zero for subtracting. If all pixels of the RGB image follow this rule of odd sum of RGB values, our classifier considers this image to have a watermark. Once it is detected, the Central Monitor is notified, and it checks whether the given RGB image is supposed to be at that network node. We assume that it is possible to determine the identity of the node that hosts a public directory. Network nodes, participating in data exchanges, use X.509 certificates, that identify their roles (e.g. Doctor or Insurance Agent or Researcher). Also, identity can be based on node's IP address or other attributes. For instance, if a data obligations database (see Table 2) has no record that an 'Unknown' recipient is allowed to access a patient's medical data, a leakage alert will be raised. Instead of applying such a simple conversion function F (r, g, b) to every pixel such that the sum r + g + b is odd, more secure conversion functions F can be used.

3.1.3 Proposed Work

The following additional data leakage detection/prevention methods [23] are proposed:

- **Partial data disclosure**
  (a) Authorized client after the first data request is only given a portion of accessible data;
  (b) Monitoring the client's trust level, which is constantly re-computed by the Central Monitor using the following metrics: (a) number of sent/received data requests, (b) number of rejected data requests, (c) number of communication errors, (d) CPU/Memory usage;
  (c) Disclose the next accessible chunk of data, provided trust level is satisfactory.
- **"Fake" leakage**
  In case of detected data leakage, e.g. if the exam questions got leaked, several other "fake" versions of data, i.e. other exam questions, might be intentionally leaked to lower the value of leaked data.
- **Data classification level elevation**
  The idea is to raise the classification level for leaked data class to prevent leakage repetition.

3.2 Data Leakage Damage Evaluation

Data leakage damage is evaluated using the following information [23]:

- How malicious is the recipient of unauthorized data;
- Sensitivity of data that got leaked;

- Leakage timing
- Inference threat, which indicates whether other data can be inferred from the data that got leaked

$$\text{Damage} = \text{Kds}(\text{Data Sensitivity}) * \text{Ksm}(\text{Service Maliciousness}) * \text{F}(\text{t}) \tag{1}$$

Kds denotes data sensitivity coefficient, Ksm is service maliciousness coefficient, F (t) is data sensitivity function.

Figure 5 illustrates different data sensitivity functions. The data event (e.g. final exam) happens at time t_0. Damage from data being leaked before t_0 is high. Damage from data being leaked after t_0 either immediately drops to zero (e.g. final exam got leaked after the exam is over) or decreases linearly (e.g. newly invented cryptographic hash function, which is examined by users and attackers) or remains high (e.g. export-sensitive technology).

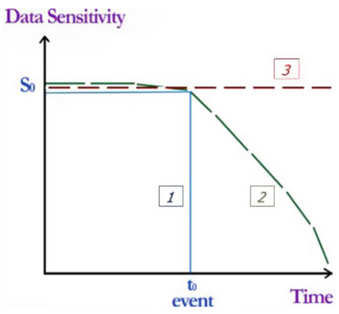


Fig. 5. Data sensitivity functions

4 Evaluation

We measured performance for clients sending data requests to EHR, represented as an Active Bundle, one per patient. Data request round-trip time (RTT) is measured between times of sending a data request and data retrieval from an Active Bundle. RTT is a sum of times spent for authentication, evaluation of access control policies and client's attributes, data leakage checks (if the data leakage detection feature is enabled), key generation and data retrieval. *ApacheBench* utility (version 2.3), as well as browser developer consoles (for Firefox browser) are used for RTT evaluation. Details of framework implementation are covered in [6].

Experiment 1
In this experiment, we aim to measure the latency of a data request sent to EHR, which is hosted by the Hospital Server, located in the same network as the requesting client. The Hospital Server that hosts EHR has the following characteristics:

Hardware: MacBook Pro, Intel Core i7 CPU @ 2.2 GHz, *16 GB memory*
OS: macOS Sierra 10.12.6.

The client sends a request for Patient ID to the EHR, represented as an extended Active Bundle, and to the basic Active Bundle, which supports neither attribute-based access control nor tamper-resistance. In contrast, EHR supports tamper-resistance and extended attribute-based access control with checking cryptographic capabilities of the client's browser [1]. The Basic Active Bundle, as well as EHR, contains four access control policies. The access control matrix for EHR is shown in Table 1. Basic Active Bundle and EHR are running on a Hospital Server, located in the same network as the client. We measure RTT for data request processing at the server side, and do not consider network delays between client and server in this experiment. Results in Fig. 6 represent latency when a first initial data request is sent to EHR. We consider it to be a special case, since the very first request to the basic AB and to EHR (which is an extended AB) takes significantly longer to be executed due to the initial authentication phase and initial evaluation of attributes. The EHR with embedded attribute-based access control and tamper resistance imposes a 12.9% performance overhead as compared with basic AB. Tamper-resistance imposes performance overhead because the digest of an Active Bundle is validated by its kernel whenever the data request arrives. Detection of the cryptographic capabilities of client's browser and checking whether it is sufficient imposes extra overhead.

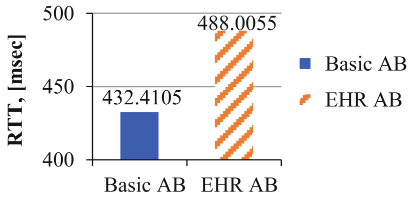


Fig. 6. EHR performance (initial request)

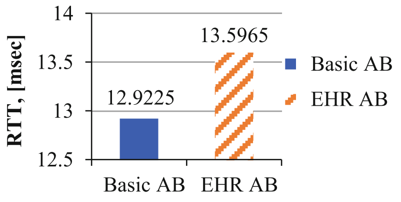


Fig. 7. EHR performance

In the next experiment, we run 50 similar data requests for Patient ID in a row. As shown in Fig. 7, mean RTT has been decreased 33.5 times for basic AB and 35.9 times for EHR. Having embedded attribute-based access control and tamper resistance in EHR imposes a 5.2% performance overhead as compared with basic AB.

Experiment 2

In this experiment, we aim to measure the latency of a data request sent to EHR which is hosted by Google Cloud Provider and has the following characteristics:

Hardware: Intel(R) Xeon(R) CPU 2.30 GHz
OS: Linux Debian 4.9.65-3 + deb9u2 (2018-01-04) x86 64, kernel 4.9.0-5-amd64
Ephemeral IP: 35.192.160.136

The procedure is same as in experiment 1, but now the client queries basic AB and EHR, running on a Google cloud instance. For the data request, we measure the overall RTT that includes network delays between client and cloud server in this experiment.

Results in Fig. 8 represent latency when the first initial data request is sent to EHR. As pointed in the previous experiment 1, we consider it to be a special case. EHR with embedded attribute-based access control and tamper resistance imposes a 2.6% performance overhead as compared with basic AB.

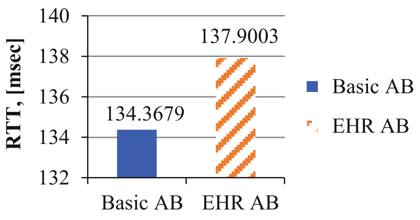


Fig. 8. Cloud EHR performance (initial request)

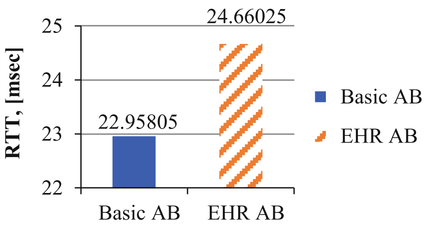


Fig. 9. Cloud EHR performance

In the next experiment, we run 50 similar data requests for Patient ID in a row. As shown in Fig. 9, mean RTT has been decreased 5.85 times for basic AB and 5.59 times for EHR. Having embedded attribute-based access control and tamper resistance in EHR imposes a 7.4% performance overhead as compared with basic AB.

Experiment 3

In this experiment, we measure performance overhead imposed by a *data leakage detection* feature added to the framework. There are four services (Hospital, Doctor, Researcher and Insurance), that are running as NodeJS servers at http://www.waxedprune.cs.purdue.edu:3000. Local data requests from a client with the role 'Doctor' are sent from the browser to the Active Bundle, hosted by the Hospital Information System at localhost: 3000. Network delays do not affect RTT measurements. The hospital server has the following specification:

Hardware: Intel Core i7, CPU 860 @2.8 GHz *x8, 8* GB *memory*
OS: Linux Ubuntu 14.04.5, kernel 3.13.0-107-generic, 64 bit

The client sends a request to EHR, hosted by the server, from *Mozilla Firefox*, version 50.1.0, browser. In our experiment, the doctor requests all available information of a patient. Active Bundles, used in Data Leakage OFF/ON scenarios, contain the same data and access control policies. The tamper-resistance mechanism and the client's browser cryptographic capabilities detection are the same and are used in both scenarios. The only difference is the data leakage detection feature. If it is enabled, it requires the Central Monitor to examine every data access before accessible data can be retrieved from the EHR, if a leakage is not detected. As shown in Fig. 10, data leakage detection support adds a 60.8% performance overhead. Before approving or denying a data request, the Central Monitor issues a SQL query to the relational database of obligations (see Table 2) in order to check whether requested data is accessible by the requesting client. Then either data request is denied or approved, and the accessible data decryption process starts. Active Bundle also contains metadata of its origin and who currently hosts the Active Bundle. In the case of a leakage alert, the hosting site will be marked as potentially malicious.

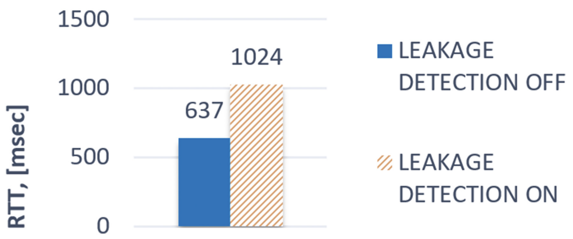


Fig. 10. Performance overhead with data leakage detection OFF/ON

5 Conclusion

We presented a comprehensive solution for privacy-preserving data exchange, which supports data leakage detection/prevention for several types of leakage scenarios. Data exchange mechanism does not need the data owner to be available since data, access control policies and policy enforcement engine are incorporated into a self-protected structure, called an “Active Bundle”. The Active Bundle mechanism provides data integrity and confidentiality, protecting data from malicious/curious cloud administrators. Active Bundle supports role-and attribute-based access control. A client’s attributes used by the data exchange model include level of cryptographic capabilities of a browser, authentication method and trust level of a client, which is constantly recomputed by a trusted Central Monitor. Data exchange also considers the context, e.g. normal vs. emergency. Data can be updated on-the-fly by multiple parties. Implemented data leakage detection mechanism imposes a 60.8% performance overhead. We also came up with a damage assessment model for data leakages. A demo video of the implemented EHR management system is available.

Acknowledgments. This work was funded by the Northrop Grumman Cybersecurity Research Consortium. The prototype was implemented in collaboration with Northrop Grumman and W3C/MIT and presented internally to Northrop Grumman in April 2017. We would like to thank Prof. Leszek Lilien (Purdue University, Western Michigan University) and Harry Halpin (MIT) for their collaboration and valuable feedback. We are also thankful to Miguel Villarreal-Vasquez, Ganapathy Mani, Rohit Ranchal and Savvas Savvides for their help and valuable feedback.

References

1. Ulybyshev, D., et al.: Privacy-preserving data dissemination in untrusted cloud. In: IEEE CLOUD, pp. 770–773 (2017)
2. Othmane, L.B., Lilien, L.: Protecting privacy in sensitive data dissemination with active bundles. In: 7th Annual Conference on Privacy, Security and Trust (PST 2009), Saint John, New Brunswick, Canada, pp. 202–213, August 2009
3. Ranchal, R.: Cross-domain data dissemination and policy enforcement. Ph.D. thesis, Purdue University (2015)
4. WSO2 Balana Implementation. https://github.com/wso2/balana. Accessed Mar 2018

5. ‘WAXEDPRUNE’ prototype demo video, part 1, https://www.dropbox.com/s/30scw1srqsmyq6d/BhargavaTeam_DemoVideo_Spring16.wmv?dl=0. Accessed Mar 2018
6. Ulybyshev, D., et al.: Secure dissemination of EHR in untrusted cloud, Project Tutorial. Purdue University (2016)
7. Lightweight data-interchange format JSON. http://json.org/. Accessed Mar 2018
8. Simmhan, Y.L., Plale, B., Gannon, D.A.: A survey of data provenance in e-science. SIGMOD Rec. **34**(3), 31–36 (2005)
9. Xu, Z.J., Wang, Z.Z., Lu, Q.: Research on image watermarking algorithm based on DCT. J. Procedia Environ. Sci. **10**, 1129–1135 (2011)
10. Ranchal, R., Bhargava, B., Angin, P., Othmane, L.B.: Epics: a framework for enforcing security policies in composite web services. IEEE Trans. Serv. Comput., 1 (2018)
11. Liu, Q., Safavi-Naini, R., Sheppard, N.: Digital rights management for content distribution. In: Proceedings of Australasian Information Security Workshop, pp. 49–58 (2003)
12. Windows media DRM. https://en.wikipedia.org/wiki/Windows_Media_DRM. Accessed Mar 2018
13. Nickolova, M., Nickolov, E.: Hardware-based and software-based security in digital rights management solutions. Int. J. Inf. Technol. Knowl. **2**, 163–168 (2008)
14. Othmane, L.B.: Active bundles for protecting confidentiality of sensitive data throughout their lifecycle. Ph. D. thesis, Western Michigan University, Kalamazoo, MI, USA, December 2010
15. W3C Web Cryptography API. https://www.w3.org/TR/WebCryptoAPI/. Accessed Mar 2018
16. Web authentication: an API for accessing scoped credentials. http://www.w3.org/TR/webauthn. Accessed Mar 2018
17. ‘WAXEDPRUNE’ prototype demo video, part 2. https://www.dropbox.com/s/4wg3vuv52j4s16v/NGCRC-2017-Bhargava-Demo1.wmv?dl=0. Accessed Mar 2018
18. Finding or verifying credit card numbers. http://www.regular-expressions.info/creditcard.html. Accessed Mar 2018
19. Nevase, J., Chougale, P., Shewale, S., Bhosale, P.: Data leakage detection. Imperial J. Interdisc. Res. **3**(5), 1232–1236 (2017). http://www.imperialjournals.com/index.php/IJIR/article/view/4923/4733
20. Stamati-Koromina, V., Ilioudis, C., Overill, R., Georgiadis, C.K., Stamatis, D.: Insider threats in corporate environments: a case study for data leakage prevention. In: Proceeding of 5th Balkan Conference in Informatics, pp. 271–274 (2012)
21. Kaur, K., Gupta, I., AK, Singh: Data leakage prevention: e-mail protection via gateway. J. Phys: Conf. Ser. **933**(1), 012013 (2018)
22. Gupta, I., Singh, A.K.: A probability based model for data leakage detection using bigraph. In: 2007 Proceedings of the 7th International Conference on Communication and Network Security, pp. 1–5. ACM (2017)
23. Bhargava, B.: Secure/resilient systems and data dissemination/provenance. NGCRC Project Technology Final Report. CERIAS, Purdue University, September 2017
24. Sabadra, P., Stamp, M.: The MediaSnap© digital rights management system. In: Proceedings of Conference on Computer Science and its Applications, San-Diego, California (2003)