Developing a Secure Distributed OSGi Cloud Computing Infrastructure for Sharing Health Records

Sabah Mohammed, Daniel Servos, and Jinan Fiaidhi

Department of Computer Science, Lakehead University,
Thunder Bay, ON P7B 5E1, Canada
{sabah.mohammed, dservos, jfiaidhi}@lakeheadu.ca

Abstract. Cloud Computing has become an emerging computing paradigm which brings new opportunities and challenges to overcome. While the cloud provides seemingly limitless scalability and an alternative to expensive data center infrastructure, it raises new issues in regards to security and privacy as processing and storage tasks are handed over to third parties. This article outlines a Distributed OSGi (DOSGi) architecture for sharing electronic health records utilizing public and private clouds which overcomes some of the security issues inherent in cloud systems. This system, called HCX (Health Cloud eXchange), allows for health records and related healthcare services to be dynamically discovered and interactively used by client programs accessing services within a federated cloud. A basic prototype is presented as proof of concept along with a description of the steps and processes involved in setting up the underlying security services. Several improvements have been added to HCX including a Role-Based Single-Sign-On (RBSSO).

Keywords: Cloud Computing, Distributed OSGi, Cloud Security, Electronic Healthcare Records.

1 Introduction

"Cloud Computing" has become a popular buzzword in the web applications and services industry. Cloud computing provides a large pool of easily usable and accessible virtualized resources [1]. These resources can be dynamically reconfigured to adjust to a variable load, allowing for optimum resource utilization. Recent cloud offerings from vendors such as Amazon, IBM, Google, Oracle, Microsoft, RedHat, etc., have created various public cloud computing services where organizations are no longer required to own, maintain or create the infrastructure or applications that power their business and online presence. Cloud computing provides the potential for those in the healthcare industry, including patients, physicians, healthcare workers and administrators, to gain immediate access to a wide range of healthcare resources, applications and tools. For hospitals, physician practices and emergency medical service providers, the lowered initial investment and the elimination of IT costs offered by cloud computing can help overcome the financial barriers blocking the wide adoption of EHR systems [2]. Along with potential cost savings and scalable infrastructure, cloud computing brings with it new security and privacy issues that

need to be addressed. When utilizing a cloud based platform, potentially sensitive information must be transmitted to and stored on a cloud provider's infrastructure. It is left to the cloud provider to properly secure their hardware infrastructure and isolate customer's processing and storage tasks. This transfer of trust may be acceptable in most cloud use cases, however, in industries that must comply with data privacy laws such as PIPEDA [3], HIPA [4] and HIPAA [5], allowing sensitive information to be processed or stored on a public cloud may not be directly feasible. This article continues work on the Health Cloud Exchange (HCX) [6] system to provide security measures for protecting shared health records (EHRs) over the cloud. In particular, this article presents a lightweight Roll Based Single-Sign-On (RBSSO) authentication service for authenticating users securely on our HCX cloud infrastructure. Our RBSSO prototype extends the Apache CXF reference Distributed OSGi (DOSGi) implementation to operate securely on a cloud shared by potentially untrustworthy cloud users and providers.

2 Sharing EHRs over the Cloud

Sharing EHRs over the Internet countenances two major obstacles to break the barrier of the isolated digital silos and to enable interoperability: (1) availability of unified EHR specification and (2) the availability of a suitable centralized and secure collaboration infrastructure. Both constraints require the availability of measures and standards.

2.1 EHR Specification

Related to the EHR specifications, there are several notable EHR standards for storing, processing and transmitting patient healthcare records such as HL7 CDA [7], Continuity of Care Record (CCR) [8] and HL7 Continuity of Care Document (CCD) [9]. The CCR and CCD standards have gained wide usage and popularity among healthcare communities because of the support provided by visible IT vendors' products including Google Health[1], and Microsoft's HealthVault[2]. The CCR and CCD standards represent an XML based patient health summary which contains various sections such as patient demographics, insurance information, diagnosis, medications, allergies and care plan in a platform independent format that may be read by any XML parser. For this reason we have chosen to adopt the CCR and CCD standards as an intermediate format between EHR systems in our HCX design.

2.2 Cloud Challenges

Cloud based computing infrastructure offers organizations a pay-per-use based virtual infrastructure solution made dynamically scalable by the ability to spawn and destroy virtual machine instances on demand. This allows virtual servers to be created as

[1] www.google.com/health/
[2] www.healthvault.com

needed to meet current demand and then scaled back when strain on the system is reduced, lowering resource usage and cost. This scalability provides an ideal infrastructure for deploying a large scale but affordable system that connects end users, providers and isolated EHR systems. Adopting certain cloud computing technologies suitable for sharing sensitive information such as EHRs remains one of the challenges that vendors and researchers try to answer. Some of these security and privacy issues include:

- **Confidentiality:** Protecting cloud based storage and network transmissions from possible unwanted access by cloud providers or data leakage to other cloud users.
- **Auditability:** Maintaining logs of users' actions with the system and ensuring that no part of the system has been tampered with or compromised.
- **Security:** Preventing user credentials, which may be used for multiple services on and off the cloud from being obtained by untrusted parties including the cloud provider or other cloud users?
- **Legal:** Complying with data privacy laws that may be in effect in given geographical regions (eg. PIPEDA, HIPA, HIPAA, etc).

While many solutions for these issues exist for traditional systems, public cloud infrastructure removes control of the physical infrastructure that makes it possible to ensure a cloud provider properly secures their services and is not performing any potentially malicious activities. It may seem unlikely that large public cloud operators would intentionally violate their user's privacy, but external factors in some regions (such as legal pressure from local governments, e.g. USA PATRIOT Act[3]) may force discloser of sensitive information. Hardware based solutions, such as Trusted Platform Module (TPM)[4], that would normally provide protection for a remote system are difficult to implement in cloud environments due to instances being created on a number of physical servers that share the same hardware and lack of support from major cloud providers. Additionally, cloud computing has several challenges related to taking full advantage of the scalability gained from cloud infrastructure that limit potential solutions including:

- **Bottlenecks**: The cloud may provide seemingly limitless scalability for virtual server resources and storage, but any connections to systems outside of the cloud or lacking the same scalability quickly become a new bottleneck for the system. For example if multiple machine instances are spawned to meet an increase in demand but all connect to the same database or authentication backend provided by the same server, a bottleneck will be formed that will limit the scalability of the whole system.
- **Distributed Design**: While cloud computing is distinct from traditional distributed computing, many of the same concepts apply and must be considered in the design of a cloud application or platform. Cloud applications must be built to offer their services from multiple machine instances distributed in the same cloud rather than a traditional single server to client architecture.
- **Volatile Storage**: Most cloud infrastructure solutions (such as Amazon's EC2) do not provide persistent storage by default to their machine instances. Applications built upon such infrastructures need to take into account this static nature in their design and use additional services or solutions (such as Amazon's S3 or EBS) for permanent storage.

[3] http://en.wikipedia.org/wiki/USA_PATRIOT_Act
[4] http://en.wikipedia.org/wiki/Trusted_Platform_Module

- **Dynamic IPs**: In most cases when cloud instances are launched, a public IP address is dynamically assigned. While this may be selected from a list of static IP addresses, autonomous cloud systems are often used which automatically create and destroy instances each obtaining an unused address when initialized. This can create issues for traditional systems that expect static or unchanging addresses for servers.

3 Designing a Secure Cloud for Sharing EHRs

This section introduces our proposed architecture for securely sharing CCR/CCD patient records over a public or private cloud. The Health Cloud Exchange (HCX) system provides a dynamic and scalable cloud platform solution built on DOSGi for discovering, and providing cloud based health record services. The Role-Based Single-Sign-On (RBSSO) system provides a means of authenticating users from various organizations with these services that keeps users' credentials off the public cloud and isolated while maintaining the systems scalability.

3.1 Health Cloud Exchange (HCX)

Our proposed HCX design provides three primitive DOSGi service interfaces: EHRServices, AuditLog and EHRClient. Any number of services implementing these interfaces may run within the same cloud and are registered upon execution with a central service registry for dynamic service discovery. Multiple instances of the same service may be run simultaneously, balancing the load between them. A Service

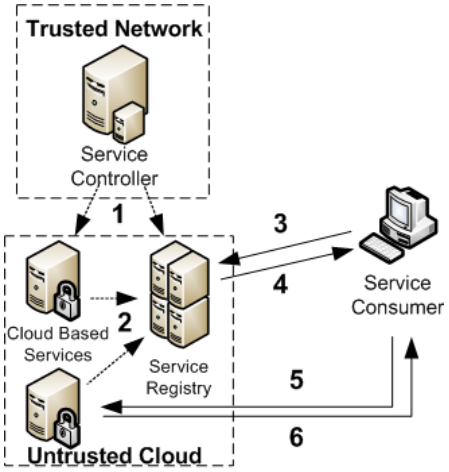


Fig. 1: An Overview of the HCX Architecture

1. Service controller starts and initializes machine instances for HCX services and the service Registry. 2. HCX servicers register them self with the service registry. 3-4. Service consumer queries service registry for a listing of available HCX services. 5-6. Service consumer sends a request to a HCX service to view, or update an EHR and receives an appropriate response. *Dotted lines* indicate interactions transparent to the service consumer.

Controller which resides in a trusted network outside of the public cloud is used to start and stop machine instances on the cloud that provide HCX services based on the current level of demand as well as load instances with their initial data. These interactions can be seen in Figure 1.

EHRServices

EHRServices are dedicated to sharing CCR and CCD formatted health records with consumers. Consumers query EHRServices for either a listing of available health records for which they have access to or a specific record using the shared EHRService interface implemented by all HCX services that share records. The EHRService interface is implemented by three main derivative services called RecordStore, EHRAdapter, and EHRBridge. RecordStores are databases of health records stored on the cloud in a relational database or other cloud storage (eg. Amazon's S3 or EBS). EHRAdapters are interfaces to existing isolated EHR systems located on the same cloud and EHRBridges are interface to external EHR systems operating outside the cloud. This allows for loose coupling between the consumers and EHRServices as consumer need only know about the standard EHRService interface and the location of the service registry to use any EHRService that becomes available on the cloud, including bridges to other systems outside of the cloud. This interaction can be seen in Figure 2.

AuditLog

The AuditLog service is used to keep and store an uneditable audit log of all actions that have been performed on the EHRs and services, including views, changes and removal of records. These logs keep a permanent record of user's actions that can be used as evidence in case an abuse of a user's credentials occurs. The AuditLog service is called directly from EHRServices (as well as any other HCX service that may

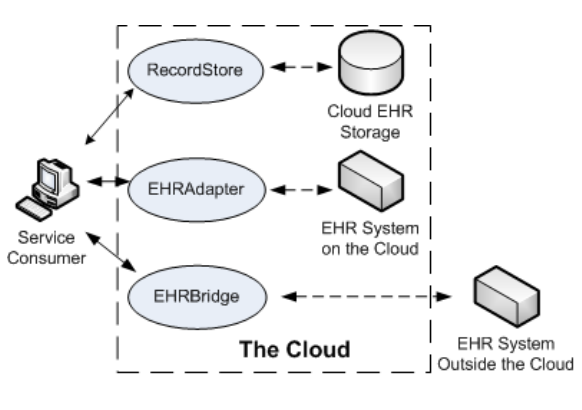


Fig. 2: **Service consumer and EHRService interactions**. *Dotted arrows* are interactions transparent to the consumer, for which all three services offer the same interface but return records from different locations.

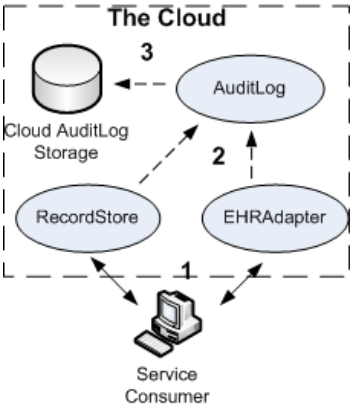


Fig. 3: AuditLog interactions. 1. Client/Service consumer makes a request on a service. 2. The service sends the RequestToken, AuditToken and request summary to the AuditLog service. 3. The AuditLog service stores a log entry for the request on cloud storage.

require a detailed audit trail) and is not accessible to normal users directly. The interface of the AuditLog has a single operation which takes the user's AuthToken, RequestToken (see section 3.2), and a summary of the users request upon the service. If the AuthToken and RequestToken are valid, the AuditLog service adds an entry to its audit log which is stored on persistent cloud storage. Figure 3 shows the interaction between an EHRService and an AuditLog service.

EHRClient

The EHRClient service is a cloud application and web interface which allows users of the system to view and update records through their browser rather than through an application implementing an EHRService consumer. The EHRClient contains both a service which controls the web interface and an EHRClient consumer which connects to other HCX record services. User authentication is still performed through the RBSSO system described in section 3.2.

3.2 Role-Based Single-Sign-On (RBSSO)

To secure HCX from access by undesired users, a roll based authentication service is required. Due to the distributed nature of the HCX architecture, traditional authentication methods are not appropriate as they would require duplication of authentication mechanism and user databases or the creation of a single point failure resulting in a system bottleneck. Additionally, users will likely need to make a request on multiple HCX services during a single session. Forcing users to provide credentials when accessing each would be an unreasonable burden, as well as inefficient if each request needed to be authenticated with a separate service. Finally it may be desired to not store a user's credentials and information directly on the cloud. A possible solution to these problems is to develop a single-sign authentication service, in which users first authenticate with a trusted party to receive an authentication token that

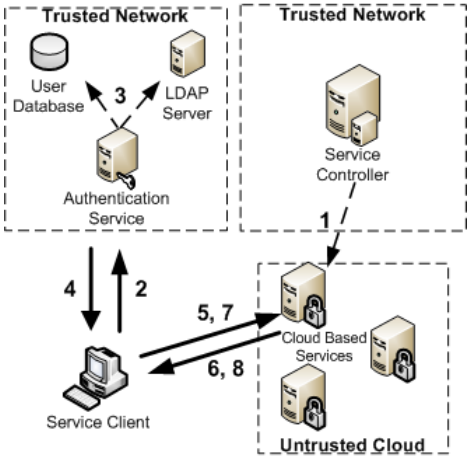


Fig. 4: RBSSO Protocol

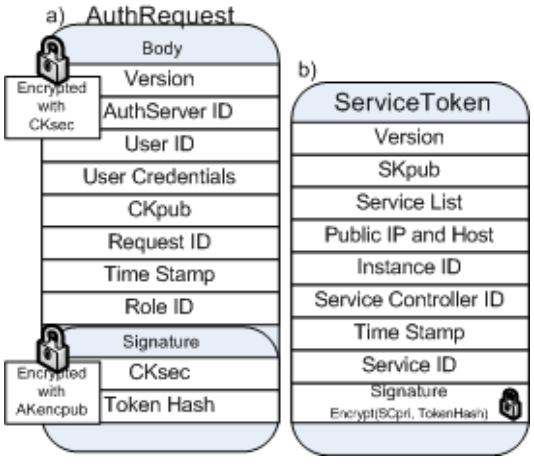


Fig. 5: (a) AuthRequest and (b) ServiceToken

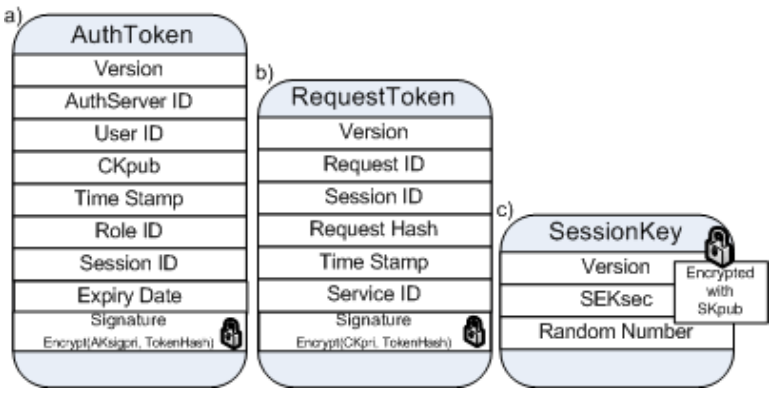


Fig. 6: (a) AuthToken, (b) RequestToken, (c) SessionKey

enables access to services that trust the same party. Several technologies currently exist which enable single-sign on capabilities, such as Kerberos [10], SAML [11], and X.509 [12]. However, the nature of the cloud and architecture of HCX make traditional solutions complicated as new machine instances are spawned and destroyed automatically based on demand and have no persistent memory to store public/private key pairs or certificates. Additionally, authentication servers become a scalability bottleneck when run outside the cloud (which may be necessary if no trusted party exists in a cloud environment). This makes any security service that have a large number of requests between services and authentication servers unreasonable or even impossible if the server is made available only to a private network to which the client is part. To solve these problems and provide user rolls which are lacking in most existing single-sign solutions, we have developed a lightweight roll based single-sign in protocol called RBSSO (Roll Based Single-Sign On) for cloud based services which may not have direct access to authentication servers. RBSSO is loosely based on X.509 single-sign on and aims to minimize the number of request on an authentication server, support a large number of authentication methods, supports sessions spanning multiple services and be relatively easy to implement and understand.

The Protocol

Figure 4 displays the interactions involved in the RBSSO protocol. Each client is assumed to be provided with the public signing (AKsigpub) and encryption (AKencpub) keys for their organizations authentication server, as well as the public signing key for the service controller (SCKsigpub). Authentication servers contain or access an organization's user credentials and are located on their private trusted networks which need only be accessible to their clients. Service controllers initialize the virtual machine instances that offer HCX services and are located off the pubic cloud. The protocol for RBSSO follows the proceeding steps, as shown in Figure 4:

1. The service controller initializes machine instances with a ServiceToken (Figure 5b), a list of HCX services the instance will provide, a list of trusted authentication servers and their set of public keys, a list of globally black listed users and the instances private key, SKpri.

2. The HCX consumer authenticates with their organizations authentication server by generating a secret key CKsec and an AuthRequest (Figure 5a). The AuthRequest, containing the user's credentials, roll they wish to activate and a public client key from the client public/private key pair created when the client program is initialized, is then transmitted to the authentication server.
3. The authentication server decrypts the AuthRequest using AKencpri and CKsec, validates the user's credentials, and checks that the time stamp and request id are acceptable. Credentials may be validated against a local database of user credentials or existing authentication infrastructure on the same trusted network (eg. LDAP).
4. Once the user is validated, the authentication server issues and signs an AuthToken (Figure 6a) with AKsigpri for the client's session with the HCX services. This transmission is encrypted with CKsec to protect the user's privacy (i.e. so the user may not be identified by outside observers).
5. Before the service consumer makes a normal request upon a service it first requests the service's ServiceToken from the instance on which it resides. The service consumer then validates the service controller's signature using SCKpub and ensures that the service is listed in the service listing and is connecting from the stated IP or hostname.
6. The consumer may now authenticate and make a request upon any HCX service on the instance by generating the secret session key SEKsec and using it to encrypt its AuthToken, the request and a newly generated RequestToken (Figure 6b) together. SEKsec is appended with a delimiter and random number and encrypted with SKpub (obtained from the ServiceToken) (Figure 6c). The ciphertexts are appended and transmitted to the service.
7. The service decrypts SEKsec using SKpri and decrypts the request, RequestToken and AuthToken using SEKsec. The service then proceeds to validate the signatures contained in AuthToken and RequestToken using AKsigpub and CKpub (from the AuthToken) and validate the fields they contain (time stamp has not expired, etc). If valid SEKsec and the AuthToken are temporarily stored for future requests with the instance until the session expires.
8. If the user has a roll active which allows the request to be performed on the service, the service complies with the request and provides the appropriate response. All further communications between the consumer and service for the length of the session will be encrypted using SEKsec. Subsequent requests on any service on the instance need only to provide a RequestToken and the content of the request encrypted with SEKsec until the session expires.

4 Implementation Details

4.1 HCX

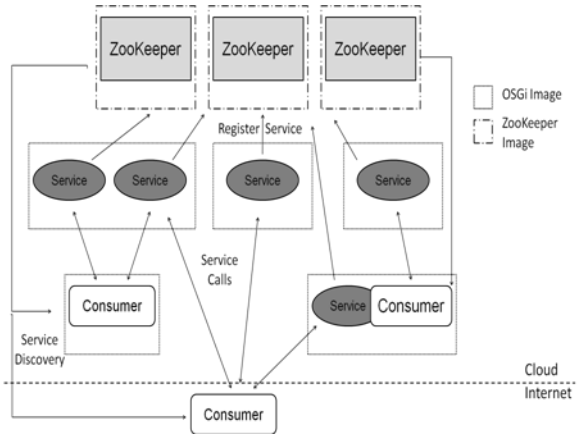


Fig. 7: Xen Images (OSGi and ZooKeeper) to Support DOSGi on the Cloud.

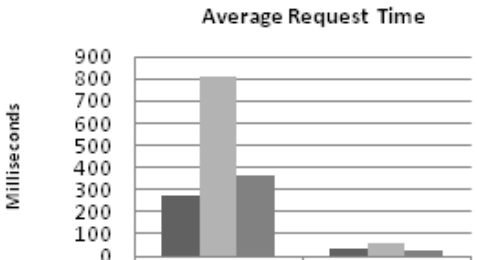
Two major open source projects were used in our implementation to provide private cloud as a service infrastructure and service discovery: Eucalyptus[5] and the Apache CXF DOSGi[6] respectively. Additionally, Amazon's EC2 and S3[7] cloud infrastructure services were used for public cloud storage and computing. Compared to other private cloud frameworks such as Nimbus[8] and abiCloud[9] Eucalyptus was chosen for its stronger community support, detailed documentation and benefit of coming prepackaged in the Ubuntu Enterprise Cloud[10] Linux distribution. Apache CXF DOSGi was chosen as it is the reference implementation for remote OSGi services and discovery. The distributed nature of DOSGi allows for a loose coupling between services and consumers through the uses of a service discovery mechanism for finding the location and type of services currently being offered in a given grouping. This is accomplished through the use of an Apache ZooKeeper[11] based cluster, in which a central service registry enables simple and scalable service look up and discovery while keeping the advantages of a distributed system (e.g. not rely on a single point of failure). Service consumers are notified of new services becoming available or going offline (a common occurrence in a cloud based setting) and are able to automatically use or discontinue use of a given service. The hardware infrastructure of the private cloud consisted of 15 identical IBM xSeries rack mounted servers connected to each other via a 1000 Mbit/s switch. Of the 15 servers, 14 were designated as Eucalyptus Node controllers which ran the Xen based virtual machine instances, while the Cloud Controller, Walrus (S3 Storage), Storage Controller and Cluster Controller services were installed on the remaining server to provide scheduling, S3 based bucket storage, EBS based storage and a front end for the cloud.

[5] http://www.eucalyptus.com/
[6] http://cxf.apache.org/
[7] http://aws.amazon.com/
[8] http://www.nimbusproject.org/
[9] http://abicloud.org
[10] http://www.ubuntu.com/business/cloud/overview
[11] http://hadoop.apache.org/zookeeper/

Adapting DOSGi for use on the cloud required the creation of two Xen based machine images. An image was required to host a standard OSGi implementation (such as Eclipse Equinox, Apache Felix or Knopflerfish) upon which the Apache CXF DOSGi, HCX bundles would be run to provide HCX's services. A second image was required to host ZooKeeper servers for service registry and discovery. As demand increases on a particular service additional OSGi machine instances may be run to load balance request between multiple instances of that service. As demand increases on the service registry, more ZooKeeper machine instances may be run to add additional ZooKeeper servers to the cluster. This set-up is shown in figure 7.

4.2 RBSSO

To evaluate the performance of the RBSSO protocol a prototype of the Authentication Server and Client where created using standard Java TCP sockets. The protocol was expanded to include length bytes to make processing the message easier. 128bit AES encryption was used for the symmetric encryption of the AuthRequest body and AuthToken body. 3072bit RSA encryption was used for the asymmetric encryption of the AuthRequest tail and the signature on the AuthToken. SHA-256 was used for generating hashes for the AuthRequest.
Two controls, SSL and Kerberos (a popular SSO system), where used to compare the performance of the protocol against. For the first control an Authentication Server was created that replaced the encryption of the body and signature of the AuthRequest with an SSL connection (the tail containing CKsec and the token hash where removed from the SSL implementation). Secondly the RBSSO protocol was also compared against the performance of a Java based Kerberos client and the MIT Kerberos 5[12] implementation which retrieved a ticket granting ticket and a service ticket (somewhat equivalent to an AuthToken in RBSSO). The performance of all three protocols (measured in average time per request) was measured on both a private isolated local



	Internet	LAN
RBSSO	276.7162	36.8437
SSL	809.5823	59.7964
Kerberos	367.0858	29.0673

Fig. 8: Average time (in milliseconds) requried to complete and verify an authencation request using each protocol. Based on 10,000 requests.

[12] http://web.mit.edu/kerberos/

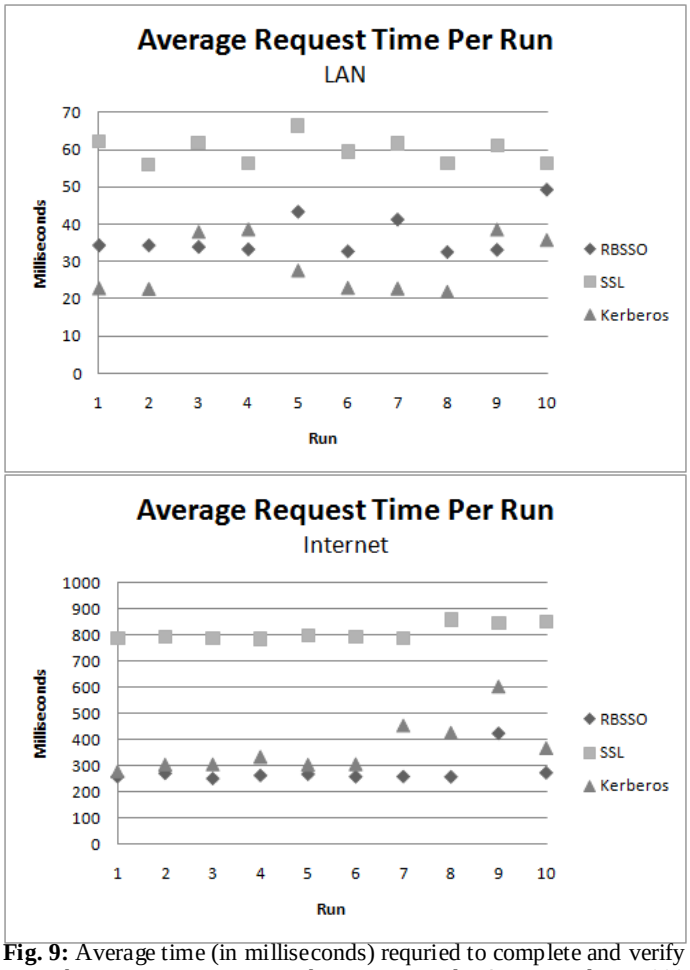


Fig. 9: Average time (in milliseconds) requried to complete and verify an authencation request over the internet and LAN. Based on 1000 requests per run.

area network and over a noisier internet connection. Each protocol was tested with 10,000 authentication requests for each network in sequential runs of 1000 requests. The results on these tests are shown in Figures 8, 9 and 10.

The RBSSO protocol performed approximately 38% faster on average than the SSL implementation on the local area network and 66% faster over the internet connection. This is likely a result of the decreased number of request involved the RBSSO protocol (no handshake is required and only a single request in made containing the AuthRequest) and explains the difference between the local and internet connections (the cost per request being higher on the connection with increased latency). Similarly RBSSO performed 25% faster than Kerberos over an internet connection but performed 21% slower over a local area connection. This is also likely a result of the number of requests, Kerberos requiring a connection to both to a Kerberos authentication server and a ticket granting server before it can make a request on a service.

5 Conclusions

The HCX system described in this paper provides a distributed, modular and scalable system for sharing health records over the cloud. This system is made secure through the extension of the RBSSO protocol also presented in this paper. We showed how to build and integrate a composite application using the Apache CXF DOSGi open source framework for sharing CCR/CCD EHR records, and how the distributed roll based single-sign on can be accomplished using the presented RBSSO protocol. The developed HCX prototype comprising of composite modules (distributed across the cloud) can be integrated and function as a single unit. HCX allows adaptors and bridges to be created for existing EHR systems and repositories allowing records to be exchanged through a standard interface and CCR/CCD record format. This is accomplished by building DOSGi based services and consumers made scalable through the cloud. The RBSSO protocol allows users to sign in once with their home organization and transparently have a session open with all HCX services for which the user's rolls give them access. There are several tasks left to our future research including providing higher information and data privacy for cloud storage thus blocking access from potentially untrustworthy cloud providers, providing in-depth details of the role based components of RBSSO, protecting cloud machine instances from tampering, and fully evaluating the performance of RBSSO in realistic cloud settings.

References

[1] Vaquero, L. et. al., "*A break in the clouds: towards a cloud definition*," *ACM SIGCOMM* Computer Communication Review, Vol. 39, pp. 50-55, (2008).

[2] Urowitz, S. et al., "*Is Canada ready for patient accessible electronic health records? A national scan*," BMC Medical Informatics and Decision Making, 8(1), p. 33, Jul. (2008).

[3] PIPEDA Personal Information Protection and Electronic Documents Act, [Online] http://laws.justice.gc.ca/en/P-8.6/index.html, (2000)

[4] Ontario Statutes and Regulations, "Personal Health Information Protection Act," S.O. 2004 Ch. 3 Schedule A, (2004).

[5] 104th United States Congress, "Health Insurance Portability and Accountability Act (HIPAA)", P.L.104-191, August 21, 1996.

[6] Mohammed, S., Servos, D. and Fiaidhi, J. "HCX: A Distributed OSGi Based Web Interaction System for Sharing Health Records in the Cloud," in *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. (Toronto, Canada, August 31 – September 3)(2010).

[7] Health Level Seven International. "Health Level Seven v3.0". [Online]. http://www.hl7.org

[8] ASTM Subcommittee: E31.25, "ASTM E2369 - 05e1 Standard Specification for Continuity of Care Record (CCR)," *ASTM Book of Standards*, vol. 14.01, 2005.

[9] Care Management and Health Records Domain Technical Committee, "HITSP/C32: HITSP Summary Documents Using HL7 Continuity of Care Document (CCD) Component," *Healthcare Information Technology Standards Panel*, Version 2.5, (2009).

[10] Neuman, B.C. et. al., "Kerberos: An authentication service for computer networks," *IEEE Communications Magazine*, vol. 32, pp. 33 – 38, (1994).

[11] OASIS Open, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0 – Errata Composite," December 2009.

[12] Housley, R. et. al., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC3280, (2002).