SECURE DATA COMMUNICATION OVER MOBILE DEVICES IN HEALTH NETWORKS

by

Ashish Sachdeva

B.Tech, Uttar Pradesh Technical University, 2009

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTERS OF SCIENCE
IN
MATHEMATICAL, COMPUTER, AND PHYSICAL SCIENCES
(COMPUTER SCIENCE)

UNIVERSITY OF NORTHERN BRITISH COLUMBIA

November 2011

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-87557-5

Our file Notre référence
ISBN: 978-0-494-87557-5

Abstract

The continuous developments in the field of mobile computing have made it possible to use mobile devices for healthcare applications. These devices can be used by healthcare providers to collect and share patients' medical data. However, with increasing adoption of mobile devices that carry confidential data, organizations need to secure the data from unauthorized users and mobile device theft. When unencrypted data is transmitted from one device to another it faces various security threats from malicious code, unsecure networks, unauthorized access, and data theft. The objective of this research is to develop a secure data sharing solution customized for healthcare environments, which would allow authorized users to securely access and share patients' data over mobile devices. We identify the vulnerable locations in mobile communication network that can possibly be exploited by unauthorized users or malicious code to access the confidential data, and develop an efficient security protocol that provides end to end data protection without compromising device's performance.

To demonstrate the feasibility of our proposed data sharing architecture, a prototype customized for Point-of-Care-Testing (POCT) scenarios was built in collaboration with Northern Health, Prince George. Simulations were performed to analyze and validate our solution against the pre-defined requirement criteria.

Table of Contents

List of Tables

List of Figures

Acknowledgement

Firstly, I would like to express my sincerest gratitude to my supervisor, Dr. Waqar Haque, for his invaluable mentorship and feedback. His personal commitment, professionalism, and persistent encouragement were truly outstanding, and I will be continually grateful for his patience and generosity throughout this endeavor. He has been immensely helpful and I could not have accomplished this without his constant and unwavering aid, reassurance and support.

I would like to extend thanks to the committee members, Dr. Alex Arvind and Dr. Pranesh Kumar for the time and effort they have put into this work. The level of personal encouragement they have shown me has been paramount to the success of my research, and I am indebted to them both. I would also like to thank my friends Anthony McCann and Han Wei Chan for their valuable feedback and creative support. Their conversations and discussions helped me expand the boundaries of my thoughts and opinions about various software and hardware issues related to this thesis.

Lastly, I would like to thank my family for always remaining a pillar of support, no matter where things have taken me. This would not have been possible without their continued faith in me.

Chapter 1

Mobile Healthcare and Security Issues

This chapter provides an overview of growing mobile device industry, and its applications in healthcare. We discuss security issues involved in sharing sensitive data over mobile devices, and introduce our proposed work that efficiently overcomes these issues.

1.1 Introduction

Mobile devices such as smartphones, Personal Digital Assistants (PDA), tablets, and laptops are available at an affordable cost today. With advancements in technology these devices have become computationally powerful and can be used for accessing emails, sharing multimedia, browsing the web, and much more. The mobile device industry is growing at a rapid rate, and the average time spent by users on mobile applications (mobile apps) has surpassed web browsing [1]. Approximately 80 million smartphones were sold just in third financial quarter in 2010 [2]. Technology analysis firm Gartner predicts that by year 2013 there will be 1.82 billion smartphones alone (not including tablets), which will be higher than 1.78 billion Personal Computers (PCs) [3]. There are over 4.6 billion subscriptions of mobile applications that enable users to accomplish majority of tasks currently done on their desktop computers. For example, Salesforce mobile application [4] provides instant access

to business and client data, logs sales, accesses charts and graphs directly on the mobile device. Square [5] is another popular mobile app that enables any compatible mobile device to accept credit card payments using a portable credit card reader. Mobile banking applications are available for accessing financial information, transferring funds, and making payments.

Mobile devices have the potential to provide preeminent infrastructure for implementing healthcare applications that enable physicians, pharmacists and nurses to access patients' health data from any remote location, and share it amongst them. For example, a physician could use a smartphone or a tablet to access information about patient's health such as lab results or medical history during a home visit, or share this information with another physician at a remote location. Future developments would enable using mobile devices to be connected to medical devices such as glucose monitors and blood pressure monitors. A study from Manhattan Research found that 71% of physicians who participated in a study consider smartphones to be essential to their practice and 84% of the physicians said that the Internet is critical to their jobs [6]. Using mobile devices in healthcare increases the accessibility of patient data and greatly improves the accuracy and speed with which medical decisions can be made.

Data sharing over mobile devices offers many potential benefits to the healthcare industry. However, there are several issues that must be handled before data sharing solutions can be deployed. Mobile devices have small screen size and limited input capability, which makes designing an optimal user interface difficult. Mobile computing platforms are different than conventional desktop platforms and have limited computational and battery resources. It is challenging to design a single application that would be compatible with mobile devices using different platforms [7]. Installation of third party

applications with unknown security vulnerabilities, or accidental navigation to untrusted sites introduce security risks such as downloading of malicious code without the knowledge of the user. Furthermore, mobile devices are prone to loss and theft due to their compact and portable nature.

Around the globe, attempts have been made to improve the way in which data can be accessed in terms of location, speed, and convenience; although data security on mobile devices is still an open challenge [8]. Due to strict privacy laws that protect patient data, it is crucial that all pertinent data is secured from theft or exploitation by unauthorized users. To fight against spamware, malware or any other malicious code, a solution that currently exists is installing an antivirus application on the mobile device that detects and removes the infection. Scanning mobile devices for malicious code, or having an antivirus program run as a background application can slow down the performance of the device making it unusable for other tasks [9] [10] [11]. Simply relying on antivirus software is not enough as they provide protection against threats from malicious code, but doesn't protect against theft or unauthorized access. Even if the mobile device is stolen, there should be no situation in which a patient's data could be compromised. One possible solution to this problem is data encryption, but the encrypted data still remains available to the unauthorized person in possession of the stolen device. There is always a possibility that a brute force attack may be applied and data may be revealed.

All the existing solutions ensuring data privacy depend on encryption using strong cryptographic algorithms that use large sized keys. The problem with such solutions is high implementation complexity and computational cost that affects the mobile device's performance, making such solutions less desirable. Most solutions encrypt the entire data on the device memory, and decrypt it every time data needs to be viewed. In this situation, the

time and computation resources required for read/write operation can become excessive. Mobile device security is a crucial area of research, but unfortunately, not many solutions are available that are affordable, efficient and scalable, and almost none are available which could be directly applied to health industry. The aim of this research is to find an efficient data sharing solution that is specially customized for health industry, and can be practically applied to provide data security without compromising mobile device's performance.

1.2 Security Issues in Mobile Data Sharing Environment

The following is a general chain of events that occur when mobile devices exchange data: the sender sends the data to the server via a network connection (either wireless or wired). The receiver is then notified about new data, at which point the receiver connects to the server and retrieves the data. The security threats in sending data from one mobile device to another can be classified into three major categories based on the location of attacks [12] [13] [14].

1. **Insecure Networks**: Data must be protected during transmission. An attacker may eavesdrop on the network connection or tamper with the data, the communication channel should therefore be encrypted. It is important to provide typical cryptographic security services such as entity authentication, data authentication, and data confidentiality [14] . Using standard mechanisms like Secure Socket Layer (SSL) / Transport Layer Security (TLS) at the transport layer or IPsec at the Internet layer can safeguard the network against such threats [14] [15]. For this research, we assume that the communication channel is secure and uses SSL / TLS at the transport layer.

2. **Insecure Servers**: "The data server is most susceptible to compromise due to mismanagement, improper configuration or worse, a hacker" [13]. There could be different types of attacks on the server, like a denial of service attack, malware or virus infections, spamware, or even worse a hacker may attack and take over control of a server. The hacker can then access and modify critical data. Storing data that is encrypted can protect against such attacks. Since servers host large quantities of data, encrypting/decrypting the entire collection of data every time an access is required can be expensive in terms of computational cost [13]. Optimization techniques must be applied to reduce computational cost involved. Finding such optimization techniques is an important component of this research.
3. **Insecure Terminals**: The mobile device itself is susceptible to various threats for a multitude of reasons. These devices resemble PCs in that they utilize complex software, which invariably contains bugs and vulnerabilities. These vulnerabilities have posed serious threats by allowing attackers using Bluetooth to completely take over a device [11]. Security experts are finding growing number of viruses, worms and Trojan horses that target mobile devices. The attacks are not just theft of user's financial information but also include deletion of information, artificial inflation of bills and blocking network traffic [16]. There have been popular malicious codes such as Cabir (a worm that arrives on mobile devices running on Symbian platform and uses the Bluetooth connection to replicate itself on other devices), Skull (a Trojan horse that disables Bluetooth, Internet, and SMS communication on the mobile device), and Mquito (a virus that once installed automatically sends international SMS to inflate the monthly bill) [16]. Using a firewall, access control, or antivirus software protects the mobile device against such threats, but such solutions are too demanding in terms of computational and memory resources for

real-time detection and removal of the threats [17]. Mobile devices usually use removal media such as subscriber identification module (SIM) and micro Secure Digital (SD) cards that can store data. The combination of the portable device size and the additional removable memory capacity, create opportunities for sensitive and proprietary data to be removed from a facility and stored in an insecure fashion [18].

In addition to the above threats, the loss of encryption keys must also be handled. Assuming that organizations use encryption techniques to protect the data on mobile devices and servers, it becomes important to secure the encryption keys. If an unauthorized user gains access to the keys the data is no longer secure. It is important that the users such as ex-employees of an organization, who have had access to the critical data in past, should not be allowed to access data or the encryption keys. About 60% of the reported attacks have been from disgruntled users who are either current employees or former employees [13]. Data access privileges of such users must be immediately revoked to disable any further access to the data.

1.3 Proposed Work and Contribution

We propose the development of a low cost, secure data sharing architecture that meets Health Level Seven (HL7) standards [19] for exchanging medical data. The architecture would allow healthcare providers to securely access patients' health data using mobile devices that are registered with health organizations. It integrates with Electronic Medical Record (EMR) applications that are directly connected with existing lab delivery networks inside a healthcare system. This integration enables receiving patients' lab results directly on the mobile devices. The architecture employs standard security mechanisms such as

authentication, authorization, and encryption to protect sensitive data and overcome the security issues identified in section 1.2.

The goal of the research is providing end to end data security, which includes securing the data while it is on the server, on the mobile device, and in transit. The solution enhances existing security protocols by reducing their current complexity and computation cost. It has been designed to meet the following fundamental requirements as defined in [20]:

1. Unauthorized access to the data must be denied.
2. Lost data should be recoverable to a new mobile device.
3. Read / write operations on the data should occur in a reasonable amount of time.

In addition to the above, our solution accomplishes the following extended requirements:

4. Only authorized devices must be allowed to access data.
5. A user must be allowed to register multiple devices, and data should be synchronized across all the devices.
6. Mobile devices should be able to securely interact with the registered EMR applications, in order to make its data available wirelessly on the device.

The main contribution of this research is the data sharing architecture designed specifically for healthcare environments. It consolidates data into a single data-store located at the server side, and provides security mechanisms that ensure data privacy and integrity. Our data sharing architecture outperforms existing data sharing solutions in terms of time taken for data read/write operations (see section 4.5). By moving majority of cryptography operations on the server instead of the device, we have been able to significantly reduce the computational resources required on the mobile device. The solution uses strong encryption

and hashing algorithms that have been recognized as standards by National Institute of Standards and Technology (NIST). It integrates with an organization's existing user-management services such as Active Directory, and doesn't require purchasing additional hardware to provide security. Furthermore, the use of open standards such as XML and SOAP eliminates language or operating system dependency, and ensures interoperability between mobile devices with different platforms. Hence, the deployment cost for our solution is lower than existing data sharing solutions which are strictly proprietary in terms of supported standards and applications.

1.4 Research Methodology

We use Constructive Research Methodology to build our solution. Constructive research is a well-established practice in the field of software engineering. It aims at producing novel solution to practically and theoretically relevant problems [21]. It involves problem solving through the construction of models, diagrams, plans and strategies, with well-defined phases of research. The methodology can be summarized as follows:

1. Identify a practically relevant problem.
2. Obtain comprehensive understanding of the problem
3. Construct a solution to the research problem, and build a prototype to demonstrate the feasibility of the solution.
4. Show original research contribution and theoretical relevance
5. Examine the scope for practical usage of the solution.

Following this methodology, our approach to design a secure data sharing architecture was as follows:

1. Identify the challenges in designing and implementing a secure mobile data sharing system.
2. Architect a solution to overcome the challenges
3. Build a prototype to demonstrate the feasibility of the solution
4. Run simulations to perform thorough testing; ensuring scalability and security, and
5. Evaluate the usability of the system through feedback from general physicians

In order to identify the challenges related to our research, we studied existing literature on mobile device security, and data sharing solutions in medical and non-medical areas. Through an extensive literature review we found out the strengths and weaknesses of different approaches. We then constructed a solution that meets the requirements identified from our initial motivation (as discussed in section 1.3) and subsequent literature review. To demonstrate the feasibility of the researched solution, a prototype customized for data sharing within a health organization was built. The prototype consists of an Apple iOS [22] based mobile application that runs on any Apple mobile device (iPhone, iPod or iPad), and enables healthcare providers to gather, share, or monitor data related to patients' health. We present the literature review in chapter two, our proposed solution in chapter three, and prototype implementation in chapter four. Chapter five presents the conclusions of our research.

Chapter 2

Literature Review

This chapter provides an overview of various mobile healthcare applications, and existing data security solutions in mobile environments. We discuss the salient features of different security techniques and existing data sharing solutions. We identify their strengths and weaknesses, and summarize the common drawbacks and limitations.

2.1 Introduction

Clinical care requires healthcare professionals to be able to access patients' health data in an efficient and timely manner. Mobile healthcare has untapped potential and is still in early stages of development. Many mobile applications have been developed to aid different medical purposes ranging from general purpose applications used for broadcasting health tips, to more specialized ones for emergency or ambulatory environments. Some popular mobile healthcare applications include:

- ECG-Notes: An application to assist healthcare providers in interpreting Electrocardiographs (ECGs). It is designed to be a reference guide for physicians and surgeons, and helps in identifying any abnormalities in patients' ECG by comparing the graphs with stored diagnostic criteria. It also provides useful information that

outlines critical medications needed in a cardiac emergency along with indications, dosage, contraindications, precautions and side effects for major emergency drugs [23].

- HealthReflex: An application that enables users to track their personal and family health information by using any iOS or Android based mobile device. Users can store clinical reports, immunizations, procedures, test results and manage their medication history on the device, and share this information with a physician who can then track prescription history or diagnostic reports [24].
- BodyMedia: An application to help control obesity by using wireless sensors to track users' physical activities, calories and sleep patterns. The sensors collect more than 5000 data readings every minute that can be accessed by an authorized user such as a physician or dietician through an online website [25].
- Electronic Point-of-Care (e-POC): A PDA based electronic healthcare system that manages patients' information in an ambulatory care environment [8] [26]. "Paper based information is available to a clinician at a point-of-care scenario is effectively limited to what the clinicians are able to carry" [27]. e-POC helps removing this limitation by allowing physicians to collect and exchange medical data in real time using a PDA with Internet connection.
- mCare: A telephony-based messaging system developed by U.S. army to send messages via cell phones to wounded soldiers in the outpatient phase of their recovery [28]. Patients with mild traumatic brain injuries are the target population for mCare, and receive a minimum of six messages per week about general health tips, appointment reminders and general announcements. These messages are disseminated from a central website where healthcare provides can enter and control

message content. The messages are sent over a secure Virtual Private Network (VPN) tunnel and can be accessed by patients by entering a Personal Identification Number (PIN) chosen by the user during registration for the service [28].

In addition to above mentioned, the following prototypes have been built:

- Northern Arizona State University developed a mobile app to assist with healthcare crisis due to insufficient medical staff, facilities, and funding in tribal areas of Arizona State [29]. The application was deployed on a ViewSonic Pocket PC to collect data related to patients' diabetes while conducting a home visit. Data gathered includes a thorough analysis of the patient's feet in order to uncover potential foot wounds. Healthcare providers are able to enter notes, schedule appointments, and make referrals all of which is stored as a part of patient's record. The information resides in a relational database stored on the mobile device and can be transferred to other medical facilities over the Internet.
- The Automated Remote Triage and Emergency Management Information System (ARTEMIS) is an ongoing research which aims at improving the clinical care under emergency/disaster situations by providing real-time physiological information of injured soldiers to first commanders/responders and command personnel [8] [30]. The application monitors patients' data using a mobile device and transmits it to remote medical facilities. This allows medical facilities to initialize triage processes (process of determining the priority of patients' treatments based on the severity of their condition [31]) and efficiently deliver treatment procedures to the medic. The primary target population is the 25% of soldiers who die between five minutes to six hours of injury due to lack of efficient communication with remote medical facilities that causes delays in providing the treatment on time [32].

A common drawback with such applications is lack of measures to protect patients' data from theft and unauthorized access. Most of the times patients' data is stored as plain text locally on the mobile device, making the data open to many security risks. This discourages wide adoption of such applications within health organizations in spite of many potential benefits [33]. Mobile healthcare is an emerging area and requires regulatory standards to control thousands of available applications. Several obstacles still need to be addressed before mobile healthcare can be widely deployed. Based on different types of security scenarios discussed in Chapter 1, the work done in mobile security can be classified in two categories – securing the data (including when it is on the server and on the device), and securing the communication network. Figure 2.1 shows the classification of related work. For this research we assume that the communication can be secured by using the standard SSL protocol. We explain how SSL protocol protects data during communication in section 3.2.1.

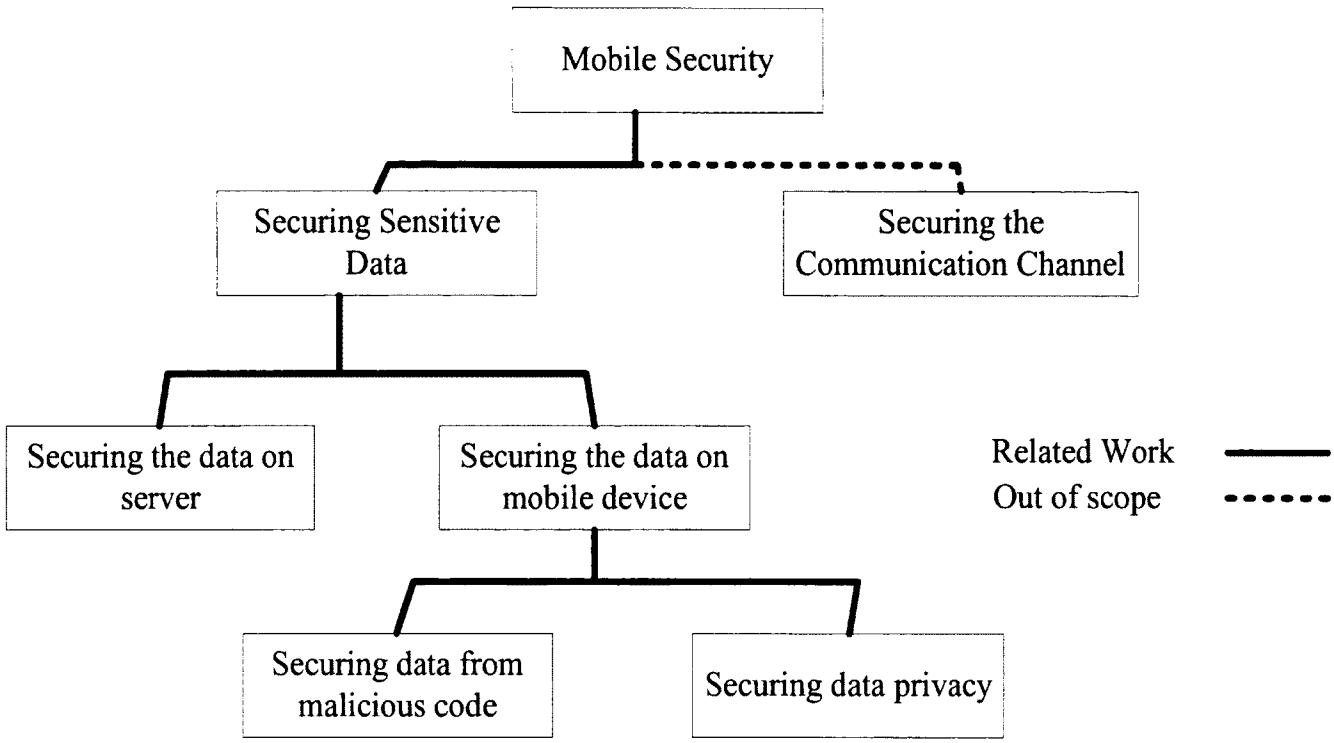


Figure 2.1 Mobile Security Classification

2.2 Securing Sensitive Data

The work done to protect the data while it resides on the mobile device can be classified into two categories: 1) Securing data from malicious code. 2) Securing data privacy.

2.2.1 Securing data from malicious code

Installing antivirus software on a traditional desktop PC is a common measure to protect data from malicious code such as spamware and malware. Such software scans the files stored on the device to identify and eliminate any malicious code by examining files for known viruses using a virus dictionary, or by identifying suspicious behaviours in programs

that may indicate infection [34]. Currently, Mobile antivirus software replicates traditional desktop model where detection services are performed on the device [10]. Due to different hardware and software characteristics the security mechanisms designed for desktop computers are not adequate for mobile devices. Antivirus software is resource intensive and is not suitable for mobile devices due to their limited computational resources. Cloud-based antivirus solutions can help reduce on-device resource consumption by moving the data (or a copy of data), and the virus detection functionality to off-device network service [10].

A cloud-based security architecture called Marvin is proposed in [11], where detection of attacks is decoupled from mobile device by placing the detection engine on a remote server (cloud). A powerful intrusion detection technique called Dynamic Taint Analysis is then used to detect different types of exploits (buffer overflows, format string attacks, double free, and so on) that change the control flow of the program. A prototype application was developed and deployed on HTC Dream / Android G1 phone platform. Figure 2.2 illustrates the architecture. The architecture has two main components:

1. **A tracer**: It resides on the mobile device and intercepts and records all signals, system calls, and read-write operations performed on the device's memory.
2. **A replayer**: It resides on the server and replays the execution trace sent by the tracer to perform a thorough analysis in order to detect any attacks.

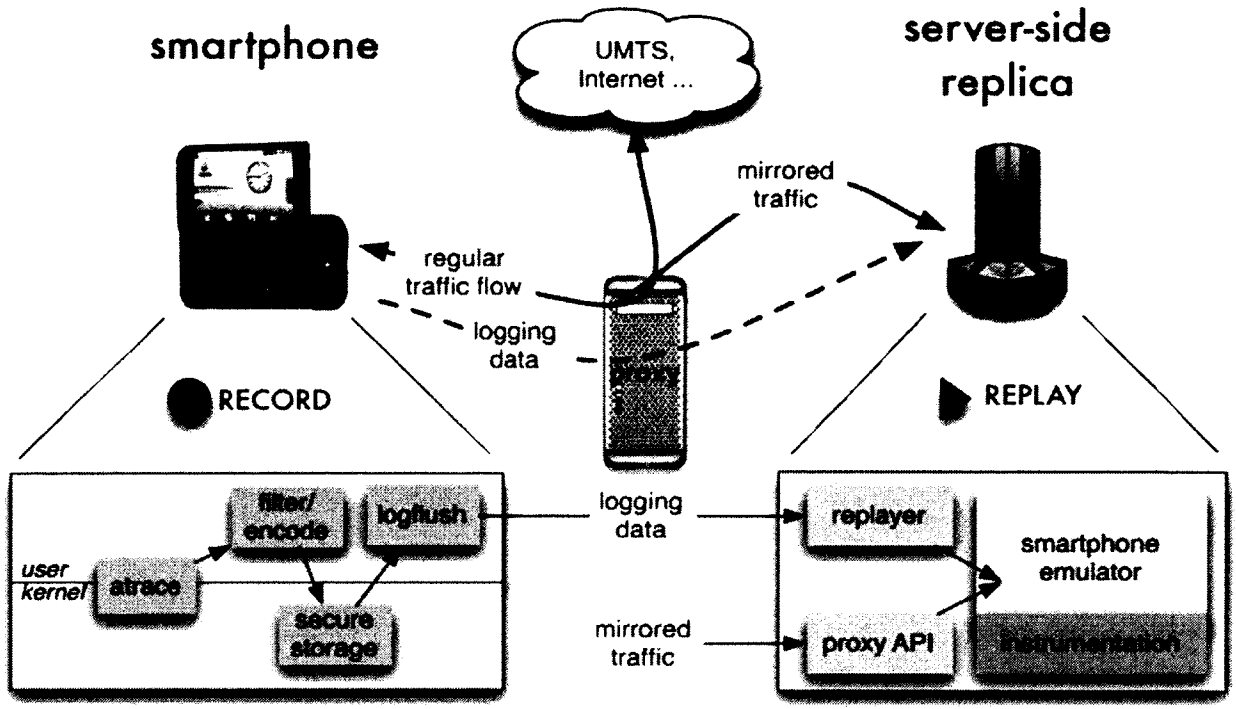


Fig 2.2 Marvin Architecture [11]

The tracer continuously monitors for any new activities on the mobile device. As soon as it records a new event, it transmits the recorded information (called trace) to the security server. The security server maintains a copy of Data stored on the smartphone, and performs a dynamic taint analysis to detect intrusions. This is achieved by re-executing the processes (as recorded in trace) on a replica of data by using a smartphone emulator. Whenever an intrusion is detected the user is informed and, depending on an organization's policy, immediate recovery procedures are started. Various options are suggested for recovery procedures such as remote locking the mobile device to prevent unauthorized access, or remote wiping the data. The strength of Marvin architecture depends on the following three main factors:

1. Location of the security server: It is assumed that the server will be hosted at a secure place.
2. When to transmit trace data: Determining the frequency of transmitting data to server is a crucial task. Transmitting data in real-time may not be necessary and the frequency of transmitting data must be optimized to conserve network resources of the mobile device. However, it should still be regular so that the delay between attack and detection can be minimized.
3. Informing the user about an attack: The user must be warned when an attack is detected, and recovery procedures must start immediately.

Since the server stores a copy of the data, Marvin provides ability to restore lost data in case of device theft or a critical attack by malicious content.

Another 'in-cloud' security model has been proposed in [10] that aim at reducing the device's C.P.U, memory, and power resources consumed by antivirus software. This is accomplished by moving the detection service from mobile device to a server. Similar to Marvin architecture, this model consists of two primary components:

1. **Host Agent**: A lightweight application that runs on mobile device and inspects file activity on the system. It acquires files and sends them to the server for analysis.
2. **Network Service**: The network service runs at the server. It receives files from the host agent and performs file analysis to detect malicious content.

Access to each file on the mobile device is trapped and diverted to a handling routine that generates a unique identifier (UID) (such as hash) for the file. The unique identifier is compared against cached UIDs of previously analyzed files [10]. When a UID is not found in the cache list, the host agent acquires the corresponding file and sends it to the server.

When network service receives the file a thorough file analysis is performed to detect any malicious content. The architecture doesn't specify recovery measures in case a threat is detected.

Benefits of cloud-based antivirus solutions

Such solutions have three prime benefits:

1. Better detection of malicious content: Since the detection service is hosted on a computationally powerful desktop computer instead of the mobile device, several detection engines can be used for file analysis enhancing the accuracy of threat detection.
2. Reduces on-device resource consumption: On-device resources can be conserved by transferring files to a server for threat detection.
3. Reduces on-device software complexity: By deploying a relatively simple software-agent on the mobile device and pushing the complex detection software on the server, the complexity of mobile software can be minimized.

Limitations of cloud-based antivirus solutions

1. Disconnected operation: The data security in such solutions always relies on a network connection. Mobile devices may enter a disconnected state where the mobile agents may not be able to effectively utilize the network-based security services. In such an event, data can be compromised.
2. Data Privacy: Such solutions require sensitive data to be sent to the server for analysis. It is important that users understand the privacy implications of such

solutions and organizations be able to enforce limitations on what data is transmitted to the security server.

Relying on antivirus software for data security is not enough. The virus definitions in software's virus-dictionary must be updated in a timely manner. The virus authors are ahead of the curve and have started writing "Polymorphic Viruses". Polymorphic viruses in part or whole encrypt or modify themselves in order to not match their definitions in a virus dictionary [34]. Antivirus software is passive in nature as it waits until an attack is detected. Delay in attack detection can cause critical security breach as the data might get compromised by the time an attack is detected. Antivirus software does not provide protection against unauthorized data access. Hence, using antivirus software alone for healthcare applications would not provide adequate security measures required to protect sensitive data.

2.2.2 Securing Data Privacy

Data privacy can be ensured by using appropriate encryption and authentication techniques. Authentication and encryption are the fundamental blocks for any security mechanism that protects data from unauthorized access and ensures data confidentiality. For this research, we need a reliable authentication technique that provides both user and device authentication. Encryption can be hardware-based or software-based. In hardware-based solutions the encryption/decryption functionality utilizes the device's high-speed physical memory, which accelerates the overall speed of encryption/decryption process. It protects the data even if the operating system is not active, for example if data is read directly from the hardware [35]. Software-based solutions do not facilitate their own dedicated physical memory. They use device's C.P.U and main memory through interfaces provided by the

underlying operating system. As stated in [35] "The security level of a software-based cryptographic module is upper-bounded by the security level of the mechanism that protects the secrecy and integrity of the memory space it uses", the operating system usually provides protection against any other applications attempting to access the memory at the same time. The strength of the memory protection is often dependent on the robustness of operating system and its being free from flaws [36]. Most cryptographic algorithms require intermediate results to be stored on a temporary memory. These results could be closely related to the secret keys, and therefore it is essential that the memory space is secure and protected from unauthorized access.

Due to dedicated physical memory, the hardware-based solutions offer higher degree of data security and faster performance in comparison to software-based solutions. However, such solutions are expensive and proprietary, which bounds users to be dependent on a vendor for products and services. This restricts interoperability between different mobile devices and restrains users from switching to another vendor without substantial costs [37]. Hardware-based solutions have been criticized for poor documentation as the details of implementation are not always published by the vendor. This leaves the user unable to fully evaluate the security of the product and potential attack methods [35]. Software-based solutions are easy to implement and do not require purchasing special hardware to perform cryptographic operations. Most application development languages offer standard cryptography libraries which make software-based solutions easier to develop and maintain. A software-based encryption solution can be used for multiple applications and purposes including message encryption and digital signatures. For this research, we use software-based cryptography solution due to its low deployment cost, high usability and interoperability between mobile devices with different platforms. We assume that the

underlying operating system of the mobile device is robust and secures the shared memory used for performing cryptographic operations. Table 2.1 summarizes the comparison between software and hardware encryption.

	Software-based Encryption	**Hardware-based Encryption**
Performance	Slow	Fast
Management support for the solutions	Easy to manage	Poor: lack of built-in management software
Application(s)	Message encryption Digital Signatures Encrypt files and folders Encrypt a disk or partition	Encrypt a disk or partition Encrypt files and folders
Implementation Cost	Low	High
Platform Independence	High	Low; mostly tied up to a specific vendor.

Table 2.1 Software versus Hardware based Encryption [42]

2.2.2.1 Mobile Data Sharing Solutions

In this section we present some mobile data sharing solutions that are directly related to this research.

Transient Authentication

Transient Authentication [38] [39] provides user-authentication by using a hardware 'token' (key-fob), such as IBM Linux watch, that must be carried by the user all the time. For authentication, user provides a password to the token, which wirelessly interacts with mobile device on behalf of the user and provides the decryption key. The data on the mobile device remains decrypted and cannot be accessed without token's interaction with the device. At first, user unlocks the token by providing a pin/password, and binds the token and the laptop. This prevents token to accept unknown key-requests from other mobile devices. After a mutual authentication between the token and the laptop over a wireless link, a session key is exchanged and the session is established for a specified time. Since the decryption keys are automatically provided by the token, user does not have to directly interact with the mobile device while accessing data. The cached data objects on the token are encrypted while the token is out of range from the laptop. Figure 2.3 illustrates the process of decrypting file contents.

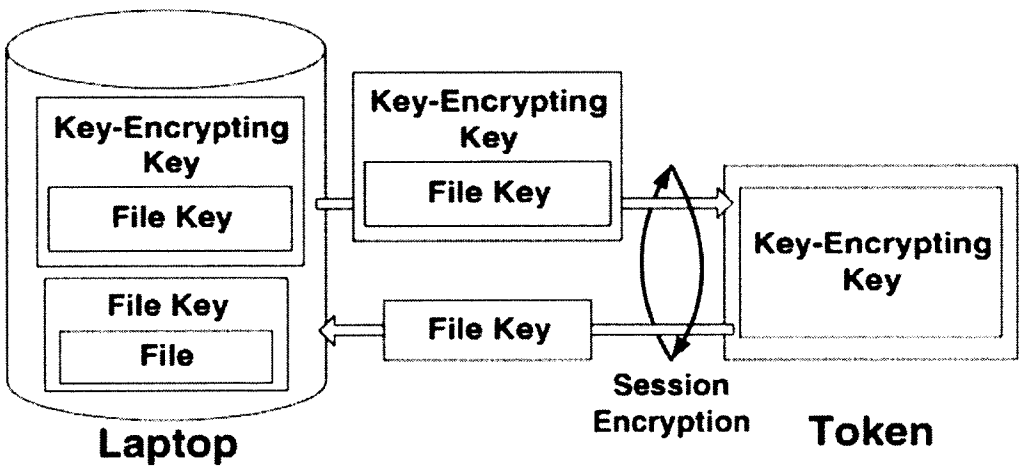


Fig 2.3 Transient Authentication [39]

Transient Authentication scheme is designed to work with laptop computers with high degree of computing power. Users potentially have multiple mobile devices which would require carrying multiple tokens, which is inconvenient. Encrypted data always remains on the device, and in the case of theft or loss an attacker can remove and inspect hard drive contents with another machine [39]. If a user loses both the token and the device, the data can be compromised. Currently, organizations are unable to audit data breaches from this event due to lack of data access tracking.

Secure Data Sharing Architecture in Mobile Environments

An approach for secure data sharing among members of a particular group is proposed in [20]. The architecture is designed to meet two key requirements:

1. In case of loss or theft of mobile device, non-group members should not be able to access data.
2. In case of loss or theft of mobile device, the original data must be recovered on a new device.

The following technologies are used to meet these requirements:

1. **Key Encapsulation Mechanism (KEM):** A cryptographic mechanism based on public key cryptosystems, used to securely exchange keys among two entities. It takes a public key as input and generates an encryption key for data.
2. **Threshold Cryptography:** It is a technique in which the private key is distributed among N number of members. When receiver needs to decrypt the data using the private key, each member computes a partial result by using their share of the

private key. All the members send their respective partial results to the receiver, who combines the results into the original message.

At first a Key Distribution Center (KDC) generates public-private key pairs using Rivest, Shamir and Adleman (RSA) algorithm [40] for the group members among whom the data has to be shared. The group's private key is divided into two shares (parts). The first share is sent to the server where the data is stored, and the second share is divided further into N+1 sub-shares where N is the number of group members. The sub-shares are distributed to each of the members and the server. Since the private key is reconstructed by obtaining more than two sub-shares, it remains protected even if some of the group's mobile devices are stolen or server's share of the key is revealed.

When data needs to be shared, a group member generates an encryption key using KEM. KEM takes group's public key as input and generates a key which is used to encrypt the data. The encrypted-key, which is generated by KEM, and the encrypted data, is sent to the server. When a group member needs to access data, a request is sent to the server for encrypted data and the encrypted-key. The server generates a partial-encrypted key by using server's share of the group's private key, and the encrypted-key. The partial-encrypted key, encrypted-data and the encrypted-key, is sent back to the member in reply to the data-access request. Subsequently, the member decrypts the encryption key by using member's share of group's private key and the information from the server. The data is decrypted with encryption key. Figure 2.4.1 and 2.4.2 illustrates the data read and write process. Table 2.2 summarizes various notations used in figure 2.4.1 and 2.4.2.

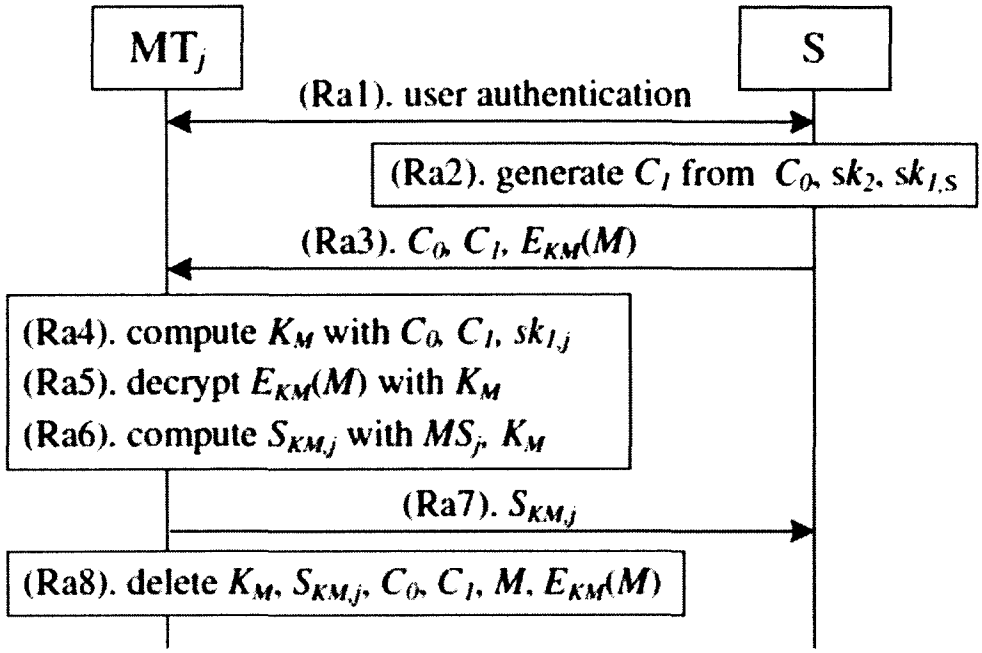


Figure 2.4.1 Read Process [20]

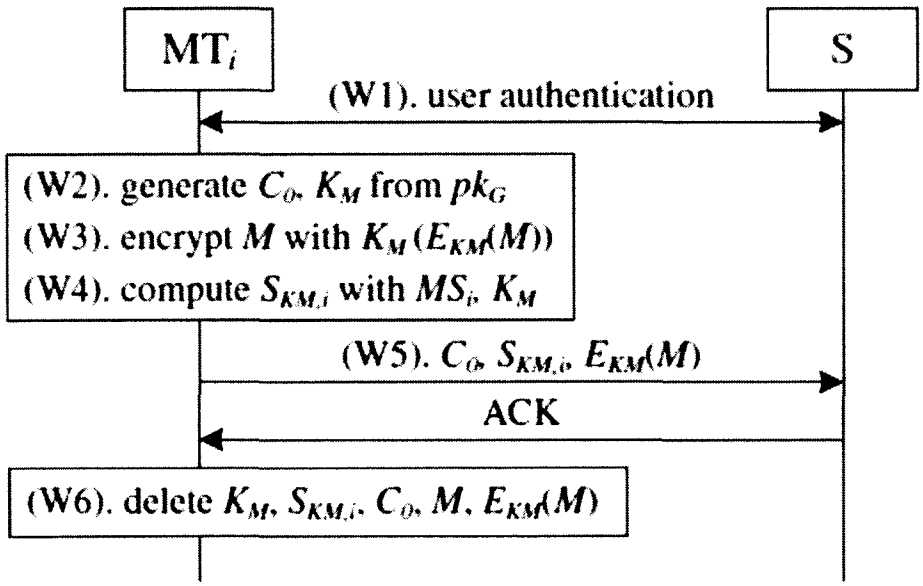


Figure 2.4.2 Write Process [20]

Symbol	Description
MT	Mobile Terminal
C_0, C_1	Encrypted Key
Km	Encryption Key
M	Data that needs to be shared
pk_g	Group's public key

Table 2.2 Notations used in figure 2.4.1 and 2.4.2 [20]

Advantages

1. Protection against lost or theft of mobile device: since the data is encrypted and resides only at the server, even if the device is lost or stolen data remains secure.
2. Group data sharing: the data can be securely shared within a group. This is an important feature in healthcare applications as patient's data might need to be shared with other healthcare providers for consultation purposes.

Limitations

1. Poor data read-write performance: Multiple encryption-decryption operations are performed when data needs to be accessed. In a group of two members, the write process takes 2.4 seconds and the read process is 3.1 seconds (for the first read). The time increases noticeably with larger data size [20]. In the healthcare scenario, with hundreds of messages sent in parallel, current times would further increase causing poor data read-write performance. Currently, the mobile device needs to perform multiple encryption-decryption operations for writing or reading data. An alternate

solution that reduces the number of times these processes on the mobile device can greatly help improving the current results. For example, sending the data as plain text over an encrypted SSL channel would provide equivalent security with faster performance.

2. Scalability and Key Management: To protect data against lost or theft of mobile device, the group's private key is divided into two parts, one of which is further divided into N+1shares where N is the number of group members. In addition, every group member generates an encryption key that is used for encrypting the data. Each mobile device must store at least two different keys and request for more keys from other group members to access data. In case one of the members loses the mobile device, all keys must be regenerated and redistributed to all group members. Managing so many different keys could be a challenging task, especially in a large group where there may be hundreds to thousands of members. If a user is associated with more than one group then it will further increase the overhead.
3. Data visibility among all group members: Shared data is visible to all participating group members. If the data needs to be shared only among few specific members of the group, a new group with different group keys must be formed. In healthcare organizations patients' data may need to be shared between few specific members only. Patient data is sensitive in nature; it should not be visible to unintended members.

Concord: A Secure Mobile Data Authorization Framework for Regulatory Compliance

Another security solution called Concord ensures data privacy by involving the organization, that owns the data, in data access process [13]. Data access requires organization's permission, and consent from an authorized user. This is implemented by

using 2-out-of-2 threshold cryptographic technique that requires at least two entities to approve data access. Mediated RSA (mRSA) cryptographic protocol is used to monitor user activities. It uses a public key cryptography where a public key is associated with two private keys. The Security Mediator (SEM) and the client get a private key each. The key to decrypt the data is constructed by using private keys of both SEM and the client. Hence, client cannot decrypt data without interacting with SEM, or vice-versa.

Concord has five main components – Trust Key Server, Connected Enforcer (C-Enforcer), Disconnected Enforcer (D-Enforcer), Data Server (DS), and the Mobile Device. Trust Key Server generates keys for other components and the data-units. The data is partitioned into various blocks referred to as data units; each unit is encrypted using data-unit keys. When data needs to be decrypted, C- Enforcer (available over wired or wireless networks) provides data-unit keys to the mobile device. D–Enforcer caches a part of the data unit keys and has the same functionality as C-enforcer; it is used only if C-enforcer is unavailable. The mobile device stores the encrypted data and a part of encryption/decryption key. A copy of data and a decryption-key is stored as encrypted at the server.

When data needs to be accessed on the mobile device, a request for decryption key is sent to C-enforcer, who decrypts the decryption-key and sends it back to the mobile device for decrypting the 'data unit key' already stored on the device. The data-unit key can now be used to access the data. If in case, C-Enforcer is not available then D-enforcer is used for acquiring decryption keys. D-enforcer is another mobile device such as cell phone or a PDA owned by the user.

Advantages

Since a user must interact with the security mediator (controlled by the organization) for data access, organizations can monitor and control access rights on the data. In the event of loss or theft of a mobile device, further requests for data accesses to previously-unread data on the mobile device can be discontinued. The organization can track down detailed information about the data that has been exposed enabling them to initiate steps for regulatory compliance [13].

Limitations

Concord requires every user to have at least two mobile devices in order to access data, as authors believe that the chances of losing both of the devices at the same time are minimal. However, if both the devices are lost or stolen the data is no longer secure. The encrypted data always remains on the device and can be brute forced in the event of loss or theft of device. Concord focuses on securing data for a single user only and doesn't supports data sharing amongst a group, which is unlikely in healthcare as patients' information may need to be shared with other healthcare providers. The computation cost and system resources required for encryption/decryption in this approach is excessive, and authors have acknowledged this limitation as well [13]. A laptop computer was used as mobile device for testing Concord's prototype. The performance results provided in [13] will degrade further if a less powerful device such as Apple iPhone, iPad or Android-based tablet is used.

A quick enhancement to Concord's framework could be using alternate threshold cryptographic solutions that would be able to reduce number of operations performed to read-write data. Alternate cryptographic solution could be using elliptic curve cryptography for encryption/decryption instead of RSA. ECC is suitable for resource constrained devices

because of smaller key size compared to traditional schemes using RSA to provide equivalent security. A 163-bit ECC provides a level of security equivalent to that provided by a 1024-bit RSA key [41].

2.3 Common Drawbacks of Existing Security Solutions

1. **High deployment cost**: Most solutions require buying special hardware, key-fobs, certificates, or even an additional mobile device that can store encryption/decryption keys and authentication information. Such solutions can be very expensive and impractical to deploy.
2. **High complexity and computational cost**: All solutions perform encryption/decryption/hashing multiple times, on the device and the server, in order to access the data. This requires dedicated system resources, and introduces additional overhead.
3. **Difficult to integrate with existing applications**: Potentially, health organizations need data sharing solution that can be embedded with their existing EMR applications and lab delivery networks. The data sharing solutions discussed in this chapter fail to provide support for this integration. Furthermore, it is essential that any security mechanism can be integrated with an organization's existing security measures. For example, most organizations use a user-management service, such as Active Directory, to strictly control authorizations and data access permissions. It is required that the new data sharing solution should provide support for such services.
4. **Scalability:** Most solutions depend on certificates for authentication, and require mobile devices and the server to store multiple cryptography keys. For example, data sharing solution discussed in (20) depends on threshold cryptography that requires

every mobile device to store at least two keys. In order to decrypt the data, a user must contact all other group members who are equally involved in decryption process, and send back the partial results to the user. A major drawback of such approaches is challenging key management. If any of the group members loses the device, all the keys for entire group must be regenerated. If a group member is not available or reachable over the network, user would not be able to decrypt the data. Similarly, other solutions such as Concord require mobile device to store multiple keys, and use them to request an additional decryption key from the server. Managing group members, participating devices, and the keys in such scenarios becomes challenging as the group size increases.

5. **Data access cannot be controlled in offline (disconnected) mode**: With the current solutions, unauthorized access to data stored on the device cannot be monitored or controlled without the network connectivity. While the device is operating in disconnected mode, the organizations would not be able to perform any preventive measures such as restricting the data access, or remotely deleting data on the device.
6. **Entire data is encrypted**: Most solutions keep the entire data and even cryptography keys as encrypted. This requires more computational and memory resources and increases the data read-write times. The data must be categorized based on sensitivity and encrypted only if required. This could reduce computational resources required, as not everything needs to be encrypted.

The National Encryption Survey held in 2006 found three most significant reasons given by the security and privacy professionals for not encrypting sensitive and confidential data to be as follows [42] [43]:

1. **System Performance**: Sixty nine percent of the participants believed using encryption would somehow affect the system performance.
2. **Complexity**: Forty four percent of the participants felt encryption techniques are too complex.
3. **Cost**: Twenty five percent of the participants did not support encryption because of additional costs involved in implementing encryption based solutions.

Hence, we conclude that all the existing security solutions are either too complex or expensive to provide adequate security measures, or they are not secure enough. Therefore, the problem of secure data sharing is still an open challenge. We propose a new data sharing solution in chapter 3, which overcomes many limitations exhibited by the existing solutions and efficiently secures the data.

Chapter 3

Secure Data Sharing Architecture (SDSA)

In this chapter, we present our solution to the research problem described in chapter 1. We first discuss important terminology and underlying algorithms that make the foundation for our work, and later we describe our solution in detail.

3.1 Definitions of Key Terms

This section outlines the important terminology used in subsequent sections of the chapter.

Authentication: It is a technique where one party proves its identity to another party. It is the fundamental block in every security mechanism that helps in distinguishing legit user from an invader. Authentication can be performed at two levels – message authentication and entity authentication. Message authentication aims at identifying the origin of message, whereas, entity authentication aims at identifying the entity itself. An entity can be a person, a client, or a server. The party who needs to prove its identity is called claimant, and the party that is trying to confirm the identity of the other party is called verifier [15].

Data Confidentiality: It refers to concealing data in a way that prevents disclosure of information. It is designed to protect data against two types of attacks – snooping and traffic analysis. Snooping refers to an attack under which the invader eavesdrops on the transmission network to intercept data and reveal its contents. A common measure to prevent snooping is encrypting data so that it is unintelligible even when it is intercepted by an attacker. However, the attacker may still intercept encrypted data and perform analysis to guess the nature of data by collecting information such as sender's or receiver's address, time of transmission, number of requests and responses. This is referred to as Traffic Analysis [44].

Data Integrity: It refers to allowing modification of data by authorized entities and through authorized mechanisms only. The data integrity can be threatened by several kinds of attacks such as modification, replaying, and repudiation [44]. Modification includes an attacker eavesdropping on the transmission link to capture data packets, and modifying its contents before it is received by the recipient. For example, an eavesdropper can capture a message from Alex to his bank requesting transfer of $100 into Bob's account. The eavesdropper may then modify the message and request transferring of $200 into Bob's account instead. Replaying attack involves intercepting messages during transmission, and resending them to the receiver. For example, an attacker may resend an intercept message requesting a money transfer of $100. Repudiation refers to a situation where the sender or receiver decline to accept that the message was sent by them. For example, the sender may refuse that he ever requested money transfer.

Encryption and Decryption: It is a technique for converting plain text into cipher text, a form that is unreadable or unintelligible to unauthorized entities. It is achieved by using an encryption algorithm such as RSA [40], and Advanced Encryption Standard (AES) [45] that

uses mathematical calculations and algorithmic schemes to perform the transformation. Encryption requires secret key(s) as input to these algorithms. The secret key(s) are known by sender and receiver, and only they can convert back cipher text into original plain text. Decryption is a technique for converting cipher-text into plain text [15].

Digital Signature: It is an electronic signature used by sender to sign the data. It is used to verify the identity of the sender and data integrity. Depending on the underlying algorithm used, the sender creates a signature on the data by using his/her private key. The signature and the data are sent to the receiver, who uses sender's public key to recreate the signature. Sender's identity can be verified by matching the two signatures [15].

Hashing: It is a mathematical concept to generate a fixed short length output by applying a hash function that performs various transformations on data. The output of this function is called message digest. It is generated in a way that given just the message digest, it is impossible to regenerate the original data. Hashing is commonly used in calculating digital signatures to provide data integrity [15].

Man-in-the-middle attack: It is an attack under which the invader eavesdrops on the private network link between the sender and the receiver to intercept messages exchanged between them. The attacker can reveal the contents of intercepted messages, and perform a replay attack by sending a same message again, or inject modified messages during the communication with receiver. Common defense against such an attack involves using data encryption and digital signatures that ensures message confidentiality and integrity.

3.2 Underlying Protocols and Definitions

We use SSL protocol to ensure secure encrypted communication of data over unsecure networks. Hashed Message Authentication (HMAC) is used for device authentication purposes. We describe both of these concepts in this section. These concepts have been summarized from [15] [46].

3.2.1 Secure Socket Layer (SSL)

SSL is a standard protocol developed by Netscape to provide security and compression services to data during transmission over networks. It ensures data confidentiality and data integrity by signing and encrypting the data received from the application layer. It uses a reliable transport layer protocol such as Transfer Control Protocol (TCP) [47], and four security protocols to accomplish a secure data transmission. The four protocols, as shown in Figure 3.1, include Handshake, ChangeCipherSpec, Alert, and Record protocol. SSL requires six cryptographic secrets/keys and two initialization vectors to establish a secure session between client and server, and exchange data.

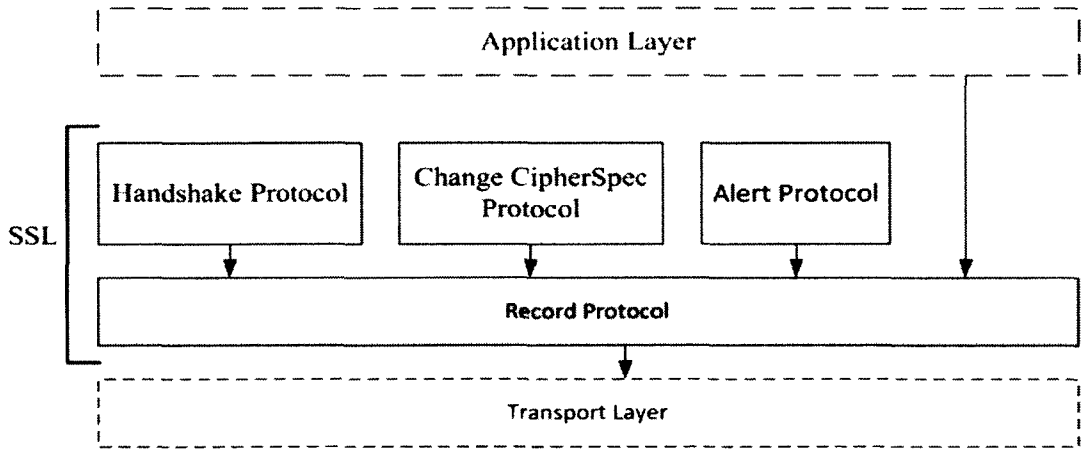


Figure 3.1 SSL Protocol Stack [15]

SSL clearly differentiates a session from a connection. "A session is an association between the client and a server", once the session is established the two entities (client and server) have common information such as a session identifier, the cipher-suit[1], and a master secret that is used to create keys for message authentication and encryption. To be able to exchange data two entities need to create a connection between them. In a session, one of the entities is treated as client and the other as server, whereas both the entities are treated equal (peers) in a connection. A session can consist of many connections, but a connection always belongs to a single session. A connection may be terminated and reestablished within the same session. When a connection is terminated, the session can also be terminated but it is not mandatory. A session can be suspended and resumed again. When an old session is resumed by creating a new connection, the two entities can skip a part of negotiation process. The master keys need not to be created every time a session is resumed. The separation of session from connection prevents the high cost of creating a master key every time a session is resumed.

In order to exchange data between two entities, a session needs to be established. This is accomplished by following a SSL Handshake protocol. During the handshake, both entities negotiate on the SSL version and various compression/encryption methods to be used. By the end of initial handshake both entities are authenticated and have shared a pre-master secret. Both entities compute a master key and other cryptographic parameters by using the pre-master secret. The ChangeCipherSpec protocol is used to notify that the entities are ready to exchange data. Once the entities are in ready state, the SSL Record protocol accepts data from the upper layer protocols and fragments it into small blocks of 2^{14} bytes or less. Each fragment of data is compressed using one of the lossless methods

[1] Cipher-suit represents the key exchange, hashing, and encryption algorithms that are being used for SSL session.

negotiated between the entities during initial handshake. Finally, each data block is individually signed, encrypted, and transmitted over network.

3.2.1.1 SSL Handshake Protocol

The objective of Handshake protocol is to negotiate the cipher-suite, authenticate client with server and server with client, and exchange other information necessary for building cryptographic secrets. Handshake is done in four phases, as shown in Figure 3.1.1.

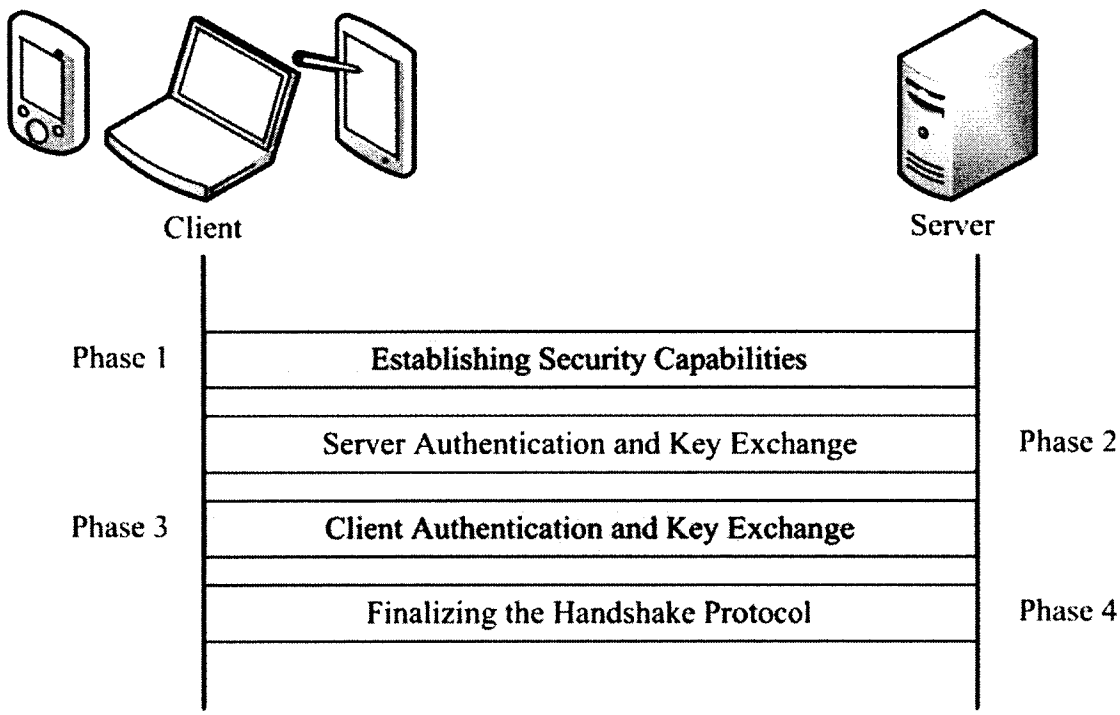


Figure 3.1.1 SSL Handshake Protocol [15]

Phase 1: Establishing Security Capabilities

During this phase, the client and the server publish their security capabilities and negotiate SSL version; algorithms for key exchange, compression, message authentication, and encryption. Finally a random number is selected by client and server each. This is used for creating a master secret. Figure 3.1.2 illustrates the steps of phase 1.

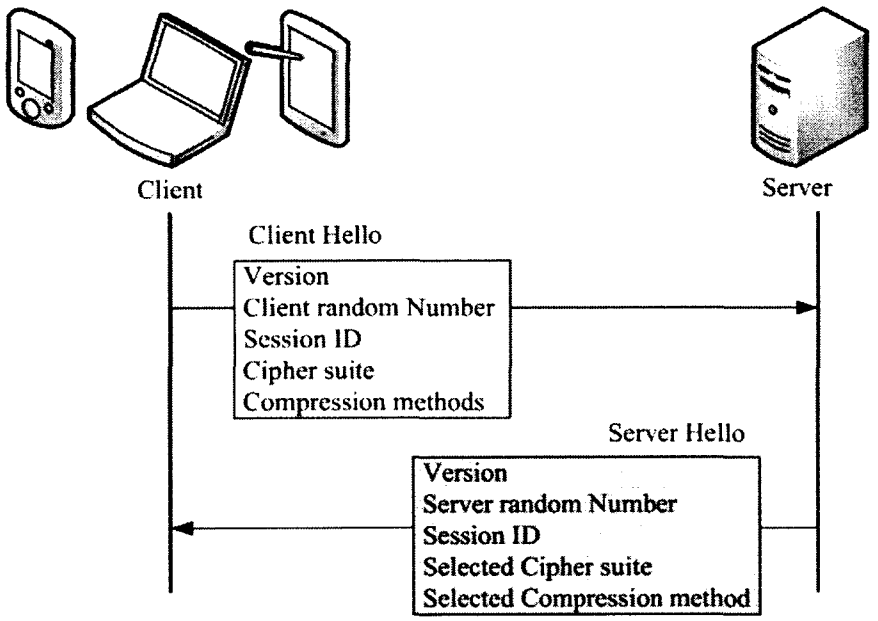


Figure 3.1.2 Establishing Security Capabilities [15]

Phase 2: Server Authentication and Key Exchange

The goal of phase 2 is server authentication. By the end of phase 2, the server is authenticated to the client and the client knows the public key of the server. The server sends list of certificates for authentication and sends its public key using the key exchange

algorithm negotiated during phase 1. The server may request client to authenticate itself by sending a message to the client requesting for its certificate. Figure 3.1.3 illustrates the steps of phase 2.

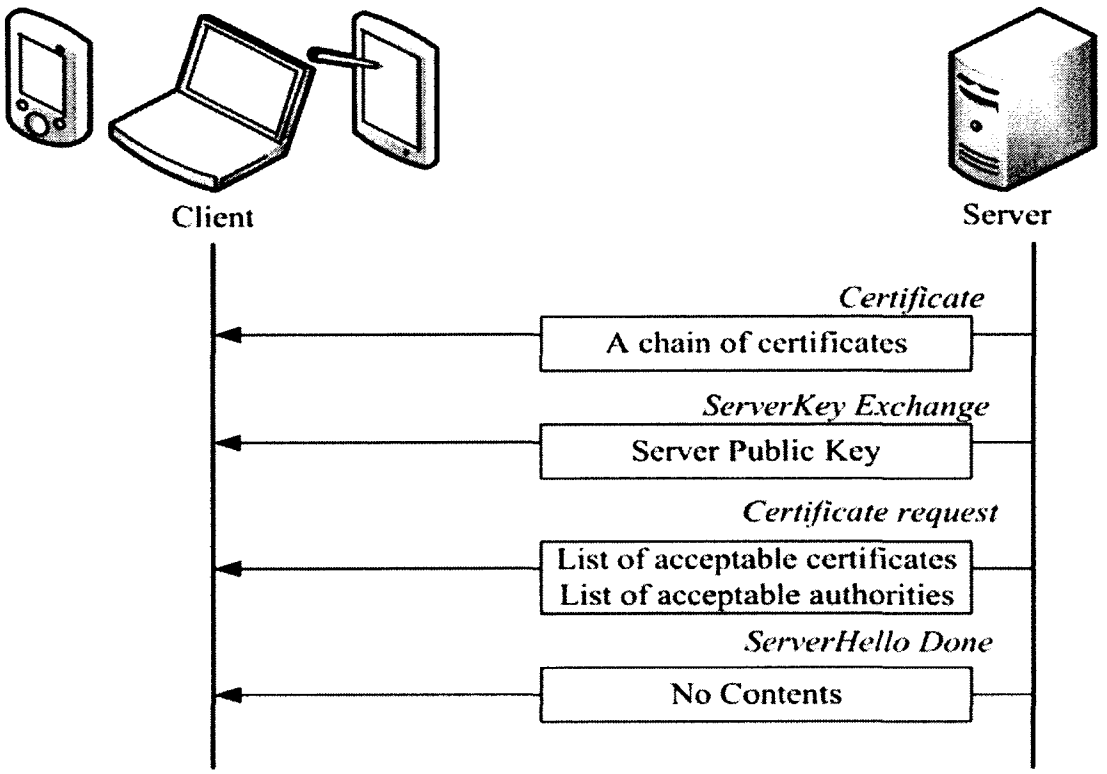


Figure 3.1.3 Server Authentication and Key Exchange [15]

Phase 3: Client Authentication and Key Exchange

This phase is similar to phase 2, it is designed to authenticate the client to the server and exchange a pre-master secret. Figure 3.1.4 illustrates the details of phase 3.

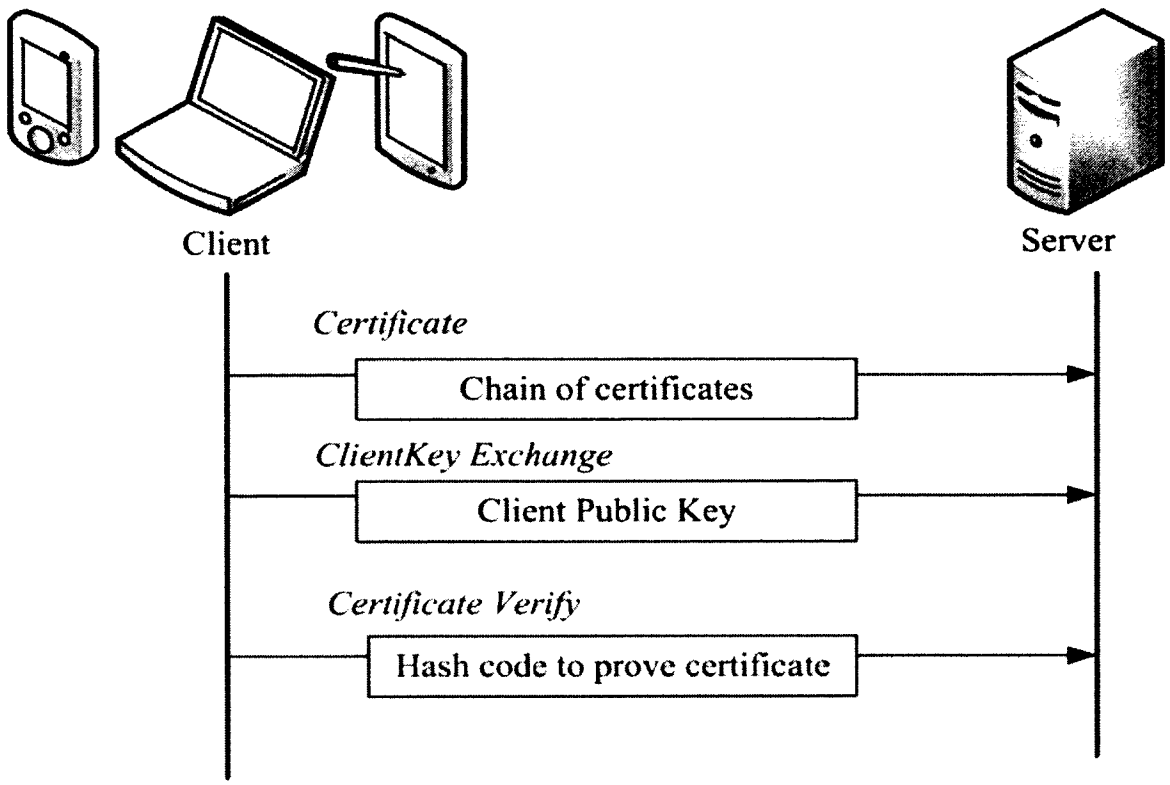


Figure 3.1.4 Client Authentication and Key Exchange [15]

Phase 4: Finalizing the Handshake Protocol

In phase 4, the client and the server change their cipher-spec from pending state to active state by using ChangeCipherSpec protocol. The client and server send a finish message to each other notifying their ready state. Both the client and the server are now ready for exchanging data. Figure 3.1.5 shows the messages exchanged during phase 4.

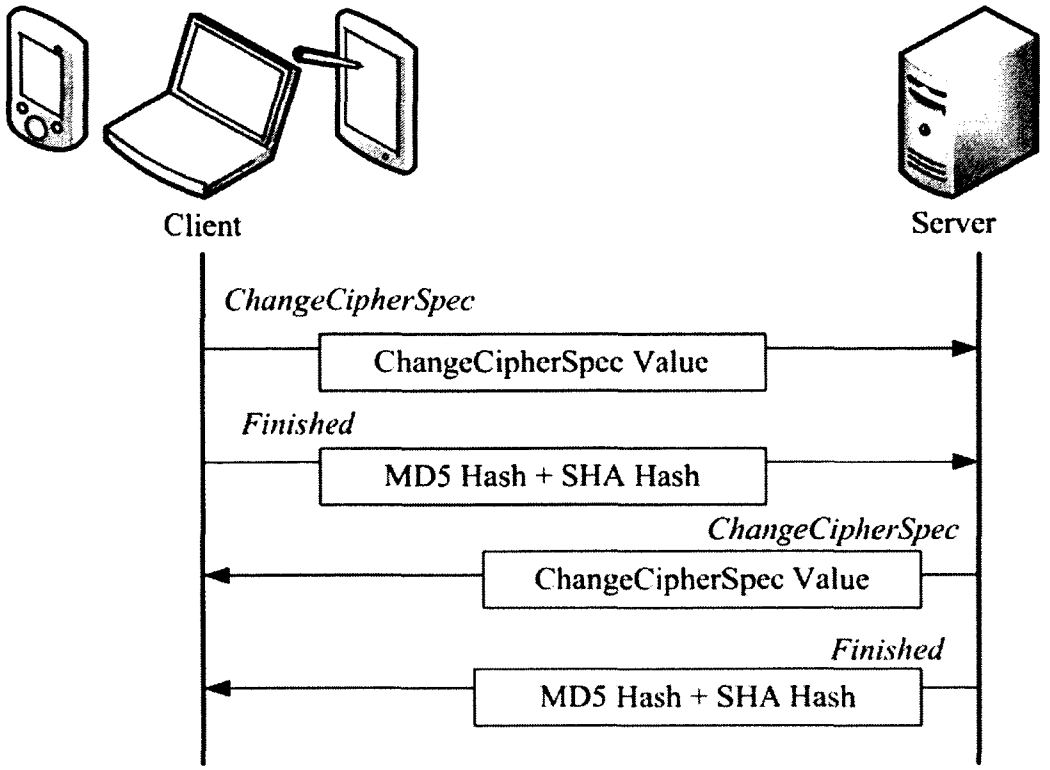


Figure 3.1.5 Finalizing the Handshake Protocol [15]

3.2.1.2 ChangeCipherSpec Protocol

When the client and the server are ready to exchange data, they need to notify each other about the same. SSL mandates that the two entities cannot use their cryptographic parameters until they receive a special message, the ChangeCipherSpec message. The message is exchanged during the Handshake protocol, and defined in the ChangeCipherSpec protocol. To keep track of parameters and secrets, the client and the server both have two states – active state and pending state. The active state represents that the sender/receiver is ready and has all the cryptographic parameters necessary to exchange data. The sender/receiver is in pending state if it is still waiting to create cryptographic parameters. In

addition, every state can have two sets of values – read (for inbound messages) and write (for outbound messages). When client needs to send the data, it moves the write (outbound) parameters from pending to active state, and informs the server about the same by sending the ChangeCipherSpec message. When the server receives the message from the client, it moves its read (inbound) parameters to active state, and sends the message back to the client. The client and server are now ready to exchange data, and can use their cryptographic parameters to sign/verify and encrypt/decrypt the data.

3.2.1.3 Alert Protocol

Alert protocol is simply used to send error notifications and report any abnormal conditions. It has only one message type called Alert-Message that describes the problem and its level [15]. Some of the problems that can be notified include no-certificate, unsupported certificate, illegal parameter, handshake failure, and decompression failure.

3.2.1.4 Record Protocol

SSL uses Record protocol to sign and encrypt messages. Record protocol accepts the data from application layer and other upper layer protocols. At the sender's end, the data is fragmented into small blocks of 2^{14} bytes with last block possibly smaller than this size. These blocks are optionally compressed, and a Message Authentication Code (MAC) is computed and padded on every message block by using the negotiated hash algorithm (MD5 [48] or SHA-1 [49]). The compressed message block and the MAC are encrypted using the negotiated encryption algorithm. Finally, the encrypted message is sent to the receiver over network using a reliable transport layer protocol such as TCP.

At the receiver end, the received message is decrypted and a MAC is re-computed on the message. Message integrity is verified by comparing re-computed MAC with the MAC

received in the message. On verifying all the received messages, the fragments are combined together to make a replica of the original message. Figure 3.1.6 shows steps performed by record protocol.

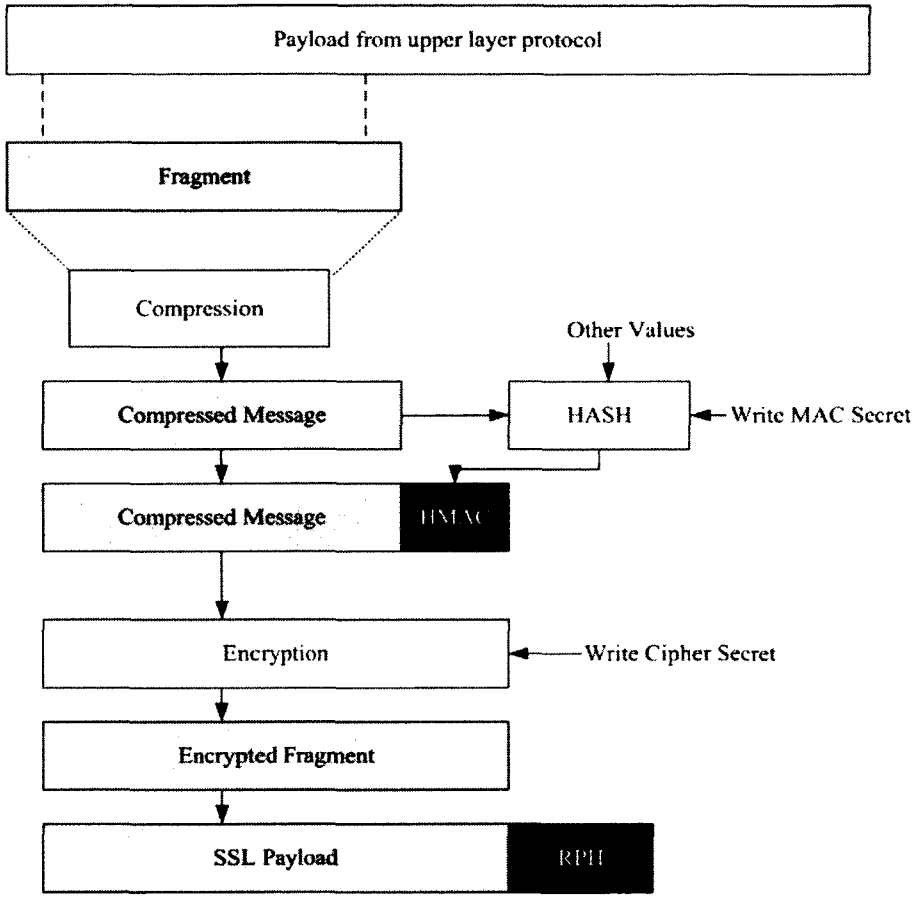


Figure 3.1.6 SSL Record Protocol [15]

3.2.2 Hashed-Message Authentication Code (HMAC)

MAC is the output of a hash function applied on messages to ensure data integrity and data origin authentication. To provide data origin authentication, MAC includes a secret, such as a secret key, which is known to the sender and the receiver only. The sender concatenates

the secret key with the message and computes a MAC using any hash function such as MD5 or SHA-1. The sender sends the message and the MAC to the receiver. At the receiving end, a new MAC is created on the message and is compared with the one received. If the two MACs match, the message is authentic and has not been modified by an adversary [15]. Figure 3.2 shows the process of creating and sending MAC.

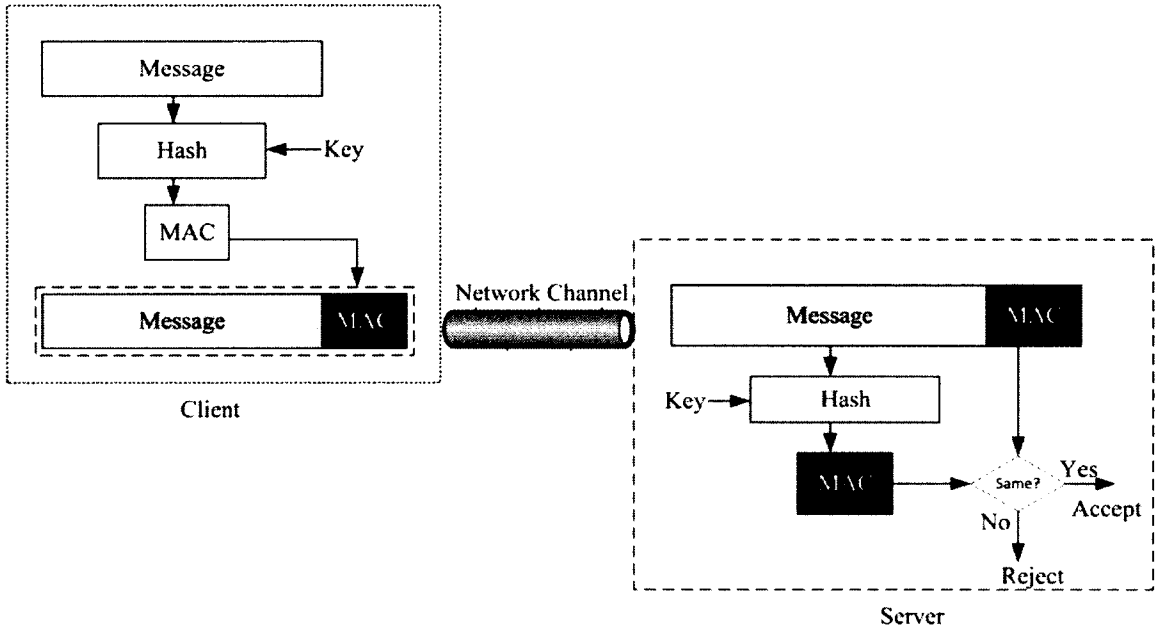


Figure 3.2 Message Authentication Code (MAC) [15]

A major issue with simple MAC is its inability to defend against man-in-the middle attack. Suppose an attacker intercepts the message and the MAC, an exhaustive search could be performed by appending all possible keys with the message to replicate the intercepted MAC. If the MAC is successfully replicated, the secret key would be revealed and any

future messages can be compromised. To prevent such situations, nested MACs were designed in which hashing is performed in two steps to increase the complexity of computing hash, and therefore increasing the cost and effort to replicate the MAC. At first, the secret key is concatenated with the message to create an intermediate message digest. Secondly, the key is concatenated with the intermediate message digest to create final digest [15].

For this research we use a standard nested MAC algorithm, called Keyed-Hash Message Authentication Code (HMAC), as issued by National Institute of Standards and Technology (NIST) for authentication and verifying data integrity [50]. As recommended by NIST, we use HMAC-SHA-256 variant that uses SHA-1 hashing algorithm with fixed 256-bit key length, which is equal to the output-length[2]. Key length is a crucial element in determining security strength of any cryptography algorithm. With HMAC-SHA-256, using a key length that is smaller than the output length can decrease security strength of the algorithm. However, due to the unique design of the algorithm key length larger than the output length does not significantly increases security strength [51]. The HMAC implementation is much more complex than simplified MAC as it includes operations such as padding and truncation to enhance security. Figure 3.2.1 shows implementation details of HMAC algorithm.

[2] Output-length refers to the size of the hash value produced by the underlying hashing algorithm [51].

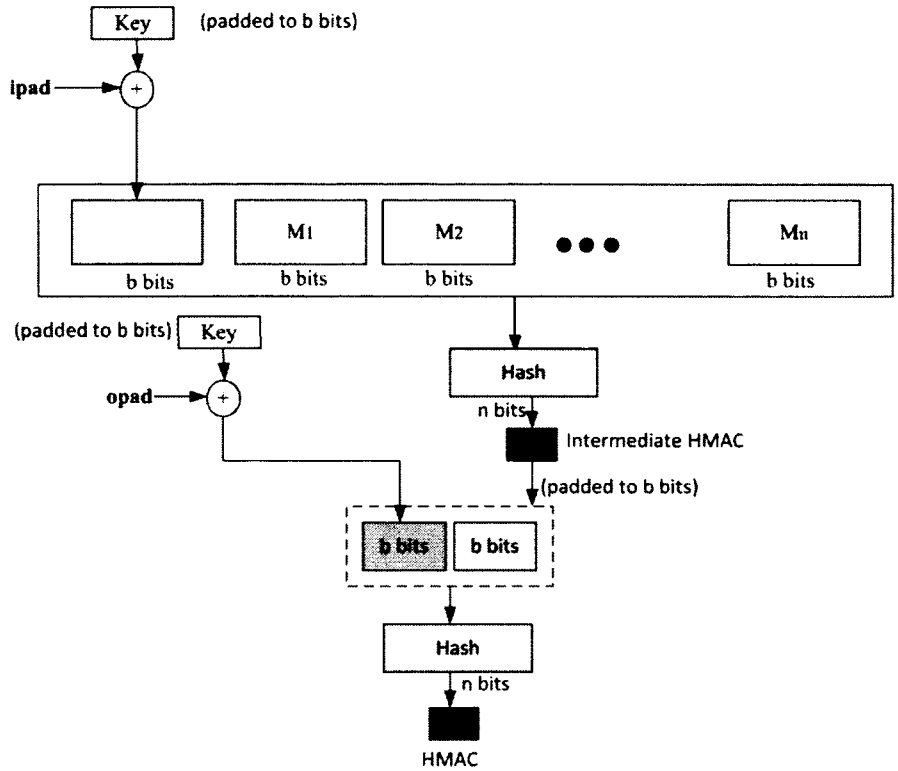


Figure 3.2.1 HMAC Implementation [15]

To generate HMAC every message is divided into n blocks of b bits each. If any block is less than b bits in length, it is padded with 0's to create b bits. The secret key is left padded if it is less than b bits in length. An exclusive-or (xor) function is applied on the secret key and a constant called input pad (ipad), to create a b-bit block. The result of the xor operation is prepended to the N-block message, and the concatenated message is hashed to create an intermediate message-digest. The intermediate message digest, also called intermediate HMAC, is left padded with 0's to make a b-bit block. An xor operation is applied on this b-bit block and a constant called output pad (opad). The result of the xor operation is hashed again to create the final n-bit HMAC [15].

3.3 Secure Data Sharing Architecture: SDSA

For this research, we have designed and developed a low cost data sharing architecture, called Secure Data Sharing Architecture (SDSA), which enables healthcare providers to securely collect, access and share patients' data using mobile devices. In order to facilitate data exchange, SDSA integrates with physicians' mobile devices, medical office Electronic Medical Record (EMR) systems, and existing lab delivery networks. Healthcare providers greatly benefit from this integration as it allows them to carry patients' health information on their mobile devices while they are out for field visits. The increase in data availability assists them in making effective decisions related to patients' treatment and improving overall patient care. Using SDSA, healthcare providers can securely share patients' medical information over mobile devices with other physicians for consultation. They can even request to receive, on their mobile devices, patients' lab results and other critical data such as medical history, prescriptions, and payment information that is stored on the EMR systems at their clinic.

In order to protect data from various security threats as discussed in chapter 1, we have designed a custom security protocol that secures communications between different components of our architecture, and protects confidential data while it is being accessed on the mobile device. The protocol is designed to meet health industry specifications (see section 3.3.2) for exchanging medical information, and makes use of strong encryption and hashing algorithms as recommended by National Institute of Standards and Technology (NIST). SDSA is platform independent allowing it to be deployed on mobile devices from different vendors and operating platforms. It supports both proprietary and open-source technologies, such as database management systems and server operating systems, which

not only increases interoperability between different systems but also helps in reducing the overall deployment cost of our solution. SDSA is scalable in nature and can be used for organizations of all sizes. It has the ability to integrate with more than one organization or clinic at the same time.

3.3.1 Architecture Components

The SDSA architecture consists of five prime components – Client application, Electronic Medical Record (EMR) application, Web Application Programming Interface (API), EMR Data Transfer Agent, and the Data Store. Figure 3.3 illustrates the architecture.

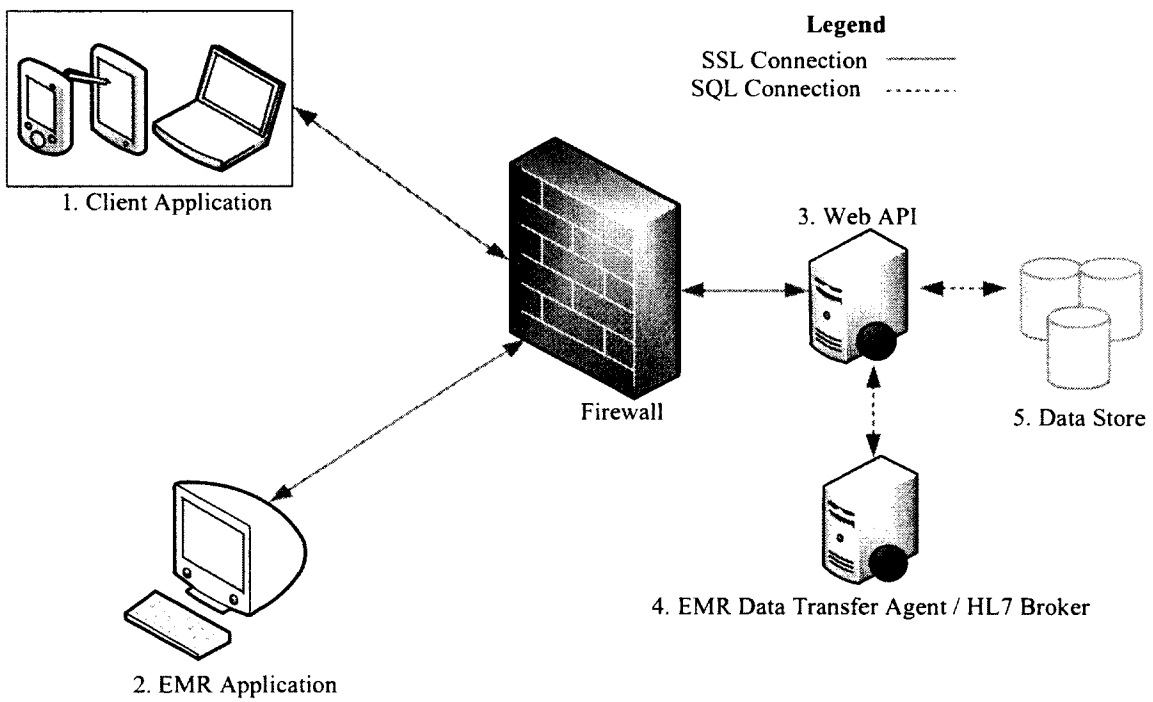


Figure 3.3 Secure Data Sharing Architecture (SDSA)

3.3.1.1 Client Application

The client application is installed on mobile devices, and communicates with the Web API in order to securely access Data stored at the server. When the client application is launched on mobile device, the user is requested for credentials to perform authentication. These credentials are transmitted to the Web API over wired or wireless network encrypted by SSL, which ensures privacy and integrity of data while in transit. Once the user has been authenticated by the Web API, data access permissions are granted for a fixed time interval. Instead of using native mobile web-browser to access data, we make use of the client application to display the results of various requests (sent to the API) to the user. This ensures that the security of our architecture is not compromised due to bugs and issues that might exist in the device's native web-browsers. The client application is responsible to perform the following tasks:

- Secure data transmission between the client device and the Web API
- Secure access to local and server-side data
- Sending/receiving of messages related to patients' health, labs and results

3.3.1.2 EMR Application

The EMR application is used by physicians to collect and store patients' medical data in a digital format. It is typically installed at hospitals and physicians' clinics, and may contain complete medical information of patients including their demographics, medical history, diagnosis, treatments, lab results and payment information. EMRs have been revolutionizing health industry for over ten years by eliminating the limitations of paper based medical records which require massive storage space, contain redundant data, and inconvenience of physically carrying and securing paper records. Computerizing medical records improves

overall efficiency in terms of storage and tracking medical history. SDSA extends the functionality of EMR application by making patient data, stored inside the application, available over mobile devices in a secure way. In order to facilitate this extension, we use an EMR data transfer agent that is responsible for extracting data from the application's database, and loading it into our Data Store located at the server side.

3.3.1.3 Web API

Web API is the backbone of our architecture. It has been built to facilitate the secure exchange of patient data among client applications and EMR systems. It is an interface that integrates with all other components of our architecture to enable secure sending, receiving and storing of data. It acts like a security gatekeeper that is situated behind a secure firewall at the server side, and allows only authorized users and mobile devices to access data, by performing various authentication checks (see section 3.4). It stores the encryption/decryption keys, and is responsible for encrypting data before it is stored inside the Data Store, and decrypting it when access is requested. If the client application needs to access data, a request is sent to the API which communicates with the Data Store, decrypts the data, and sends it to the client over an SSL connection. The API has the following functionality:

- Send/receive patients' data from client applications, and EMR Systems.
- Encrypt and store received data into the Data Store
- Decrypt the Data stored in Data Store and send it to client applications
- Authenticate users using Active Directory
- Authenticate devices using API keys
- Provide access control on the data

3.3.1.4 EMR Data Transfer Agent

The EMR Data Transfer Agent uses SSL to facilitate a secure connection, between an existing EMR system and the Web API. This connection enables extracting Data stored at the EMR system and submitting it to the Web API. For data exchange between the EMR system and the Data Store, the agent converts data into XML formatted using Health Level Seven (HL7) specifications. The agent runs as a background system service at the server side, situated behind a secure firewall, and periodically pulls data from the target EMR in order to keep Data Store updated.

3.3.1.5 Data Store

Over years, security experts have made their best efforts to consolidate data in a central repository as it is more effective and efficient to protect Data stored at a single location versus it being spread out. It is like securing the money deposited in banks by placing it inside a single locker and applying security measures on the locker. If the money is spread-out over different locations within a bank, every location would need its own security mechanisms. This not only increases the overall cost of security solution but also increases risk of theft, as there is more than one location from where the money can be stolen. With mobile devices being rapidly used to download and access confidential data, the data is scattered again. This increases data theft vulnerability as it is challenging to apply security measures on mobile devices and control data usage. Most security solutions discussed in chapter two have one fundamental problem in their approach. They all assume that the data would be secure if it is stored as encrypted on the mobile device. Storing encrypted data ensures data-confidentiality; however, if the mobile device is stolen or lost the encrypted data still remains visible to any unauthorized user that possesses the device.

SDSA consolidates all sensitive data into a single Data Store located at the server side. Since server has high degree of computational power we can easily apply strong security measures to protect the data without compromising the overall performance of the system. This also mitigates the risk of losing data from the mobile devices, and allows organizations to maintain a log of various data access activities on the Data Store. The data is temporarily stored on the mobile device for a short duration, while it is being viewed by the user, and must be secured on the device. We explain how data is secured while stored inside the data-store, and while it is on the mobile device in section 3.4.

3.3.2 Healthcare Industry Data Exchange Standards and Specifications

The organization and delivery of healthcare services is an information-intensive effort. In order to exchange data electronically in healthcare environments, it is essential to use standard data formats that ensure interoperability between different computer applications within an organization. For this research we use HL7 standard version 2.3.1, as used by Northern Health, for data exchange between various components of SDSA. HL7 is an American National Standards Institute (ANSI) accredited standard that is compatible with a large variety of programming languages and operating systems. It supports communications in a wide variety of communications environments, ranging from a full, OSI-compliant, 7-level network "stack" to less complete environments including primitive point-to-point RS-232C interconnections and transfer of data by batch media such as floppy disk and tape [52].

HL7 standard specifies the organization structure for XML messages, and enforces messages to use specific data types and field lengths. Every HL7 message has a message header segment that specifies the encoding character used. HL7 specifies few required

fields, such as Message Header Segment (MSH), Patient Identification (PID), and Observation Segment (OBX), that must be filled before a message can be exchanged. This ensures maximum interoperability when exchanging data among mobile devices and EMR applications. A sample HL7 message is shown in Appendix 1.

3.4 Securing Sensitive Data

SDSA provides full security for sensitive data, during transmission, while it is on the mobile device, and while it resides on the server. We make use of data encryption and combination of different authentication techniques to safeguard the sensitive data. Instead of storing data on the mobile devices, the data of every user is stored as encrypted in a consolidated data-store at the server side. The data is made available to its owner (user) as a service. In order to access data, the user must send a request to the Web API asking for data-access permissions. The request contains the details about what data needs to be accessed, and the credentials of the user for authentication. Once the user's identity is verified the requested data is decrypted by Web API and sent back to the client/user over SSL connection. The data is always formatted according to HL7 specifications for data exchange. Figure 3.4 illustrates the process of data access using SDSA.

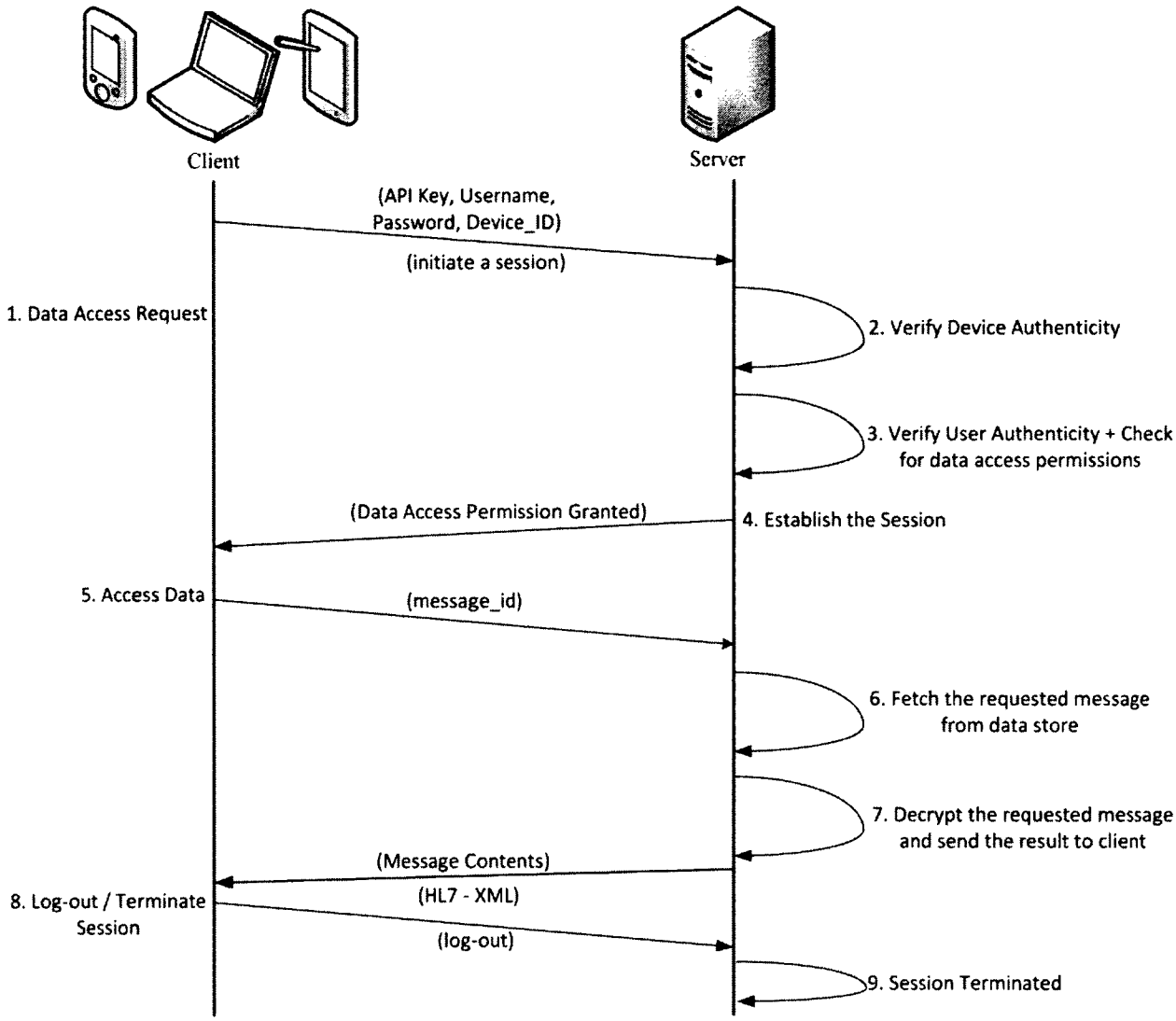


Figure 3.4 Accessing Data using SDSA

3.4.1 Authentication

In order to facilitate authentication, we have designed a two-step authentication process that performs device authentication at step one, and user authentication at second. User authentication ensures that only authorized users can access data, whereas device authentication ensures that only registered mobile devices are being used. Users must register their mobile device with the server in order to access data. Device registration is

only a one time requirement, once the device is registered users are allowed to access data by simply using their authentication credentials. Registering devices with the server, allows organizations to control data access and revoke permissions in case the device is lost or stolen.

In theory there are many ways to authenticate the user. To be authenticated the claimant (user) must identify himself/herself to the verifier (server) by using one of the following witnesses:

- Something known: A secret that is known only by the claimant, like a password, pin or a secret key etc.
- Something possessed: Something that user may have such as a token or a smart card.
- Something inherent: An inherent characteristic of the claimant such as, finger print, voice, retinal pattern, or any facial characteristics.

For our research, we use username-password (*something known*) based authentication technique due to its simplicity and convenience for users to remember a small paraphrase instead of a large size encryption key. Username-password based authentication allows integration with services like Active Directory that makes users-management easy and enables creating user-groups and hierarchies to enforce different data access permissions for different types of users [53] [54].

3.4.1.1 Device and User Registration

For user registration a new Active Directory account is created on the server where user selects a unique username and password. We assume that the organizations use Active

Directory to register users, and users are pre-verified by the organizations through their internal policies and procedures before signing up for our service. This allows organizations to have complete control on who is permitted to access data. Based on type of data-access permissions needed by the user, he/she is added to a particular domain group such as physicians, pharmacists, nurses, or administrators. Every user-group has its own unique set of permissions. Different types of permissions include read-only, write-only, and read-write data access. Categorizing users into different user-groups allows data abstraction which is important as not everyone should be able to see patients' data in its entirety. For example, a physician should be allowed to see complete medical history of patients, whereas, a pharmacist should just see patients' prescriptions.

Every user must do one-time registration for all their mobile devices that would be used for accessing data. In order to register devices, the server generates a cryptographically secure 224-bit long random number for every device [44] [55]. This random number is appended with a four digit number chosen by the user. This string, 256-bit in total, is used as a secret key that is known only to the user. We call this secret key as API key. Next, the server generates a hash on the API key by using HMAC-SHA-256 algorithm. The output of this step is a 256-bit long HMAC which is used to verify device authenticity. The server stores the HMAC only. The API key is given to the user and is permanently deleted from the server. Hence even if the server is compromised, the user keys are not revealed. The user permanently stores a part of the API key, the 224 bits or 28 bytes out of the entire length, on the mobile device, and remembers the rest 32bits or 4 bytes. Figure 3.4.1 shows device registration process at the server.

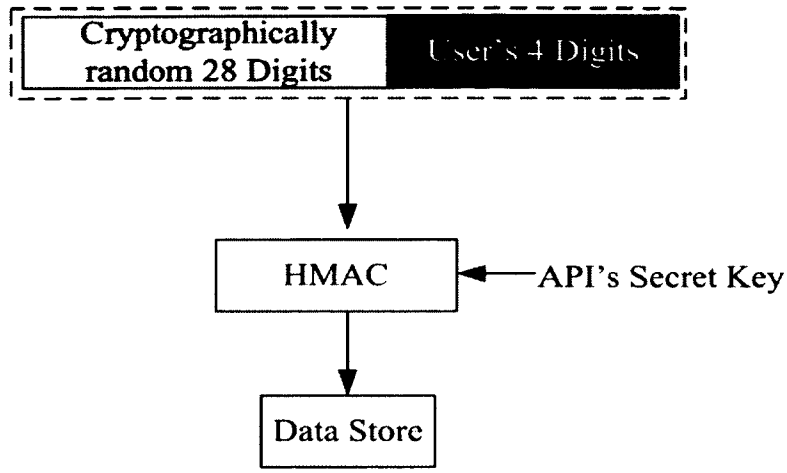


Figure 3.4.1 Device Registration at the Server

3.4.1.2 Establishing a Session

To access data on server, user needs to establish a session with the server. The user and the server undergo a handshake initiated by the user in order to establish the session. During the handshake, server verifies device's and user's authenticity, and determines type of data-access permissions. The user sends his/her credentials (username and password) along with the device's API key to the server over network using SSL connection. Device's API Key is constructed by combining the partial key stored on the mobile device, with the four digits entered by the user. On receiving this information, the server extracts the API key from the received message and generates an HMAC by using HMAC-SHA-256 algorithm, for device authentication. The new HMAC is compared with the HMAC that was stored on the server during device registration. If the two HMACs match, the device is authentic. Next, user's username and password are extracted from the message and are verified against Active Directory. Appropriate data-access permissions are given once the user and the device have been authenticated. Figure 3.4.2 shows steps of this handshake between the client and the server.

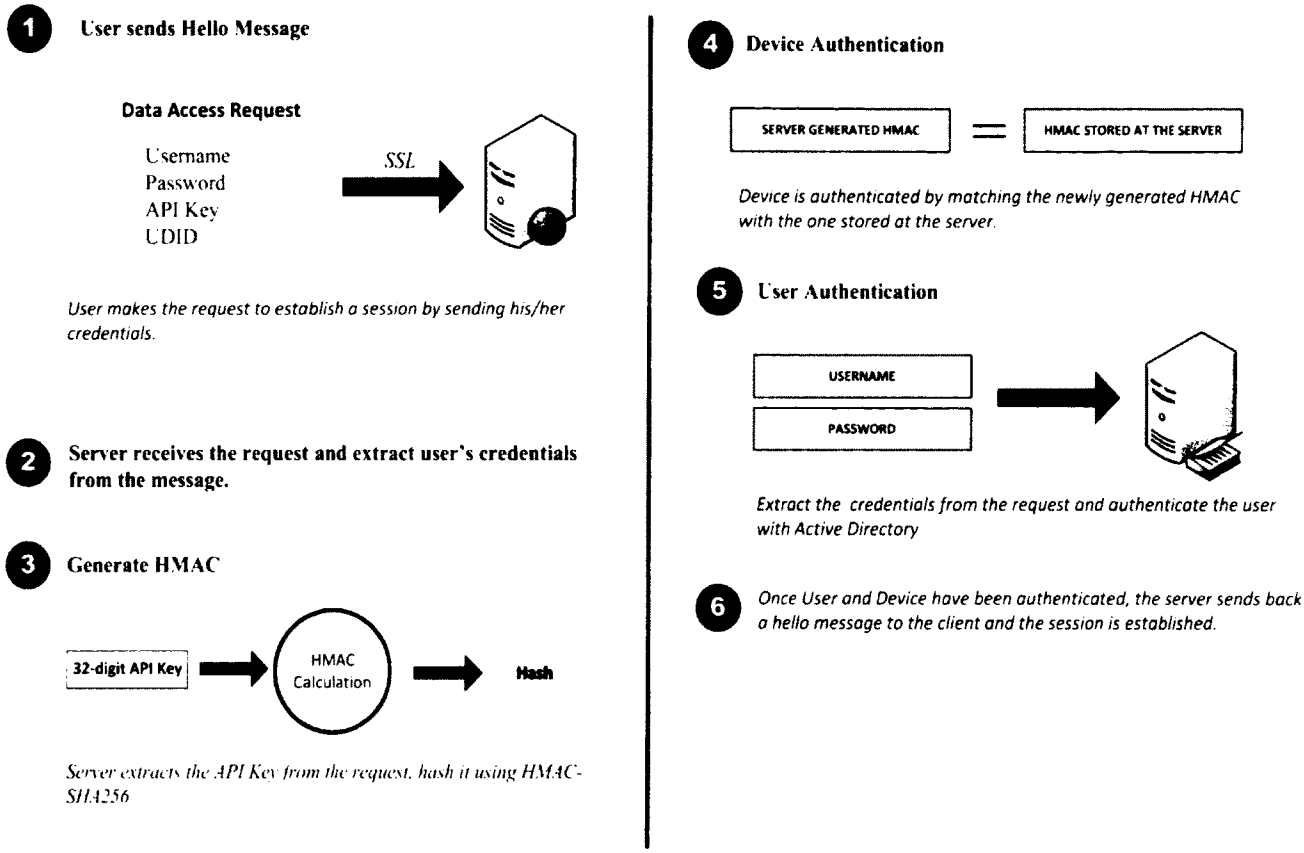


Figure 3.4.2 Establishing a Session between the Client and the Server

All the data exchanged between the server and the user is wrapped in an XML message. XML messages are portable and platform independent which allows using separate programing technology on the server and mobile device.

3.4.1.3 Accessing Data on the Server

Once the session has been established between the client and the server, user is allowed to access data until the session is terminated. Currently, data can be accessed only in 'connected mode'. Every user has a data storage space on the server; we call this storage space as user's mailbox. User can view data that resides in his/her mailbox only. Once

authenticated, user gets full access to the mailbox and can read or write new data. The mailbox can be accessed over multiple mobile devices that are registered by the user. Currently, SDSA does not support concurrent sessions between different devices and single mailbox. There can be only single active session between the client application and the mailbox. Unlike most commercial data sharing solutions such as Blackberry Enterprise Server (BES), Dropbox, and Apple's iCloud, which downloads a copy of data on the mobile device every time access is required, our solution instead allows client application to directly modify the data on Data Store itself by using various Web API calls. Since, there could be only one active session the data remains consistent and synchronized between all the mobile devices registered by the user. Figure 3.4.3 shows process of accessing and modifying data on the server.

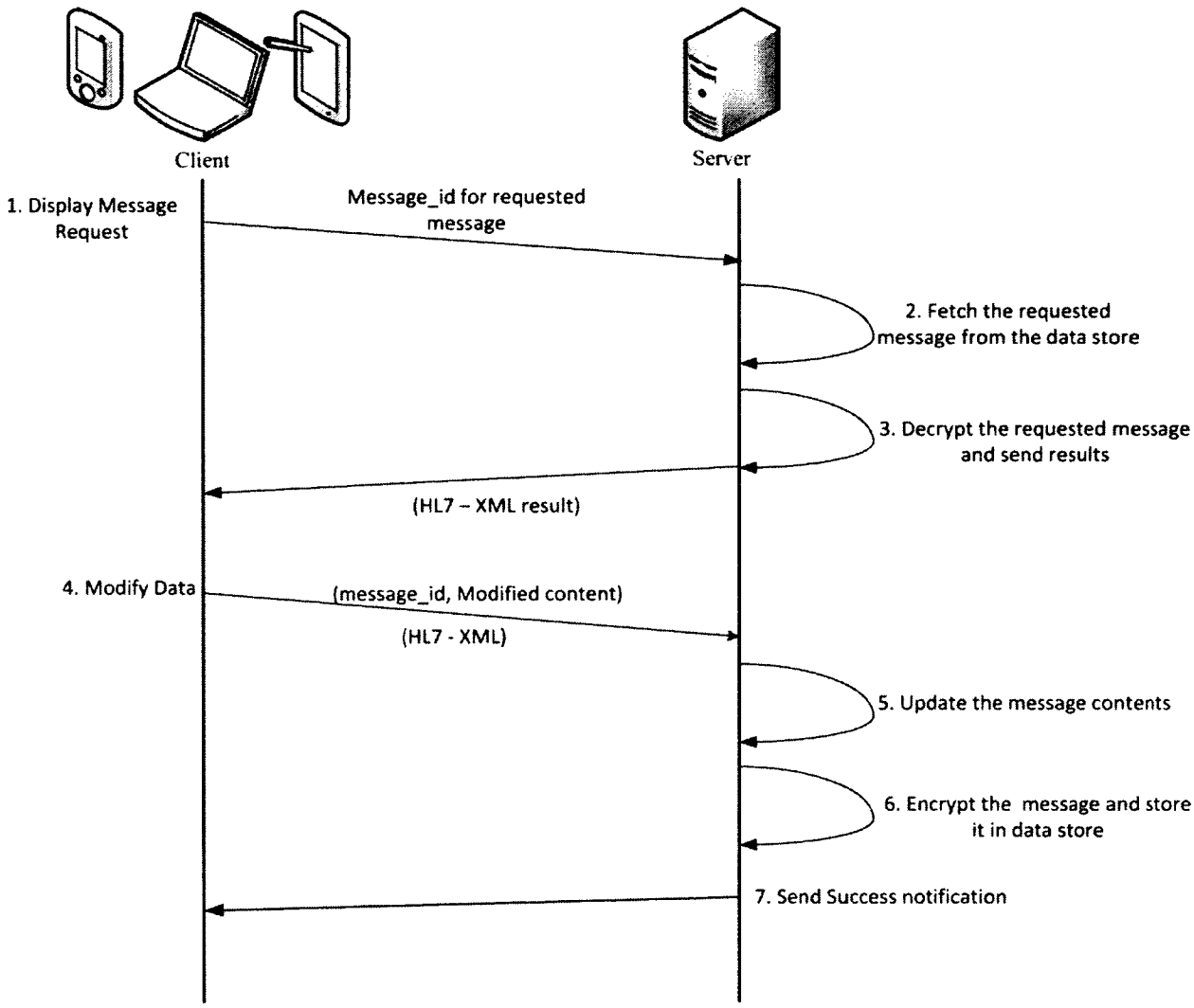


Figure 3.4.3 Data Modification Using SDSA

3.4.1.4 Exchanging Data with Other Users

A user can either create and send a new message, or forward an existing message to other users. When the message needs to be sent, a request is sent to the Web API indicating the recipient and the data that has to be sent. The server acts like a postman who takes the data from user's mailbox and keeps it in the mailbox of the recipient. In order to view received data, the recipient needs to establish a session with the server. The data can be shared among

registered users only. Figure 3.4.4 shows the process of sending a new message to other registered users. Figure 3.4.5 shows how an existing data/message can be shared among other users.

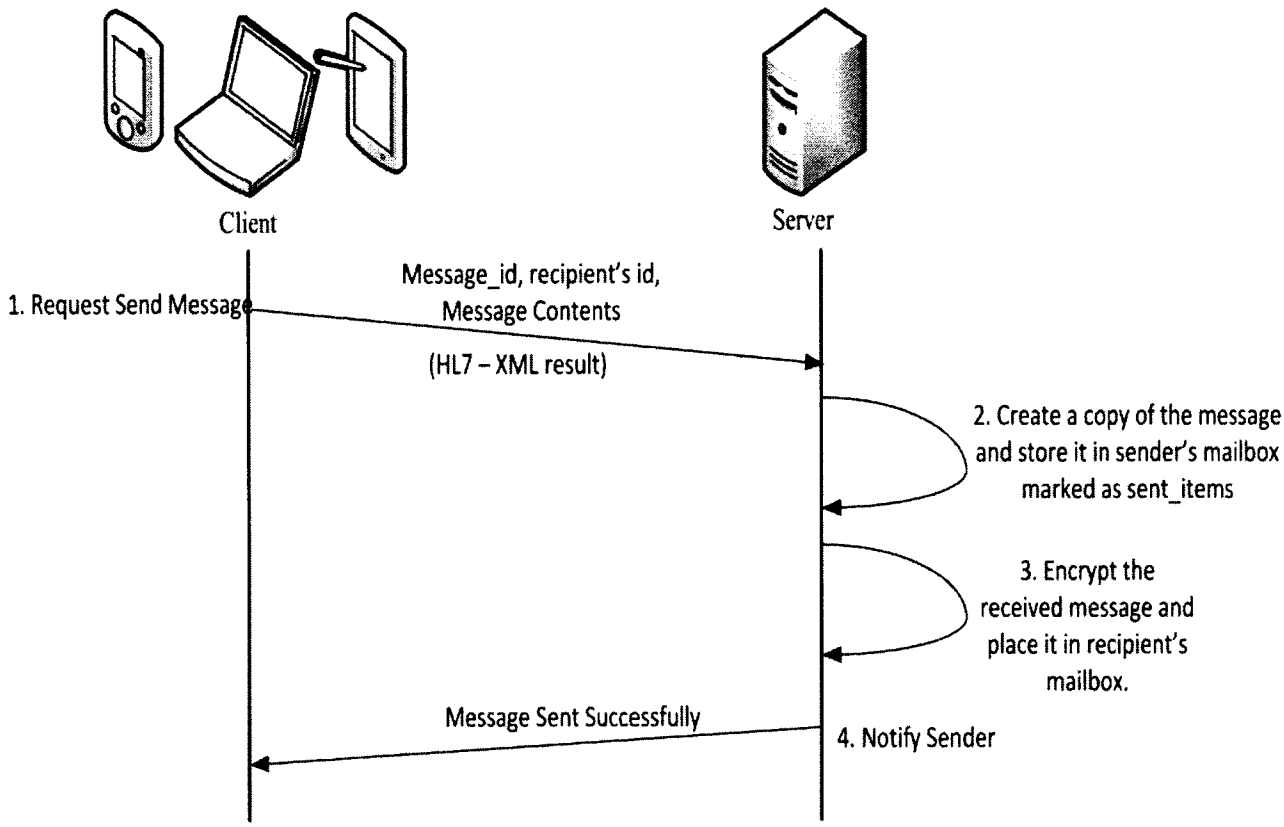


Figure 3.4.4 Sending a New Message

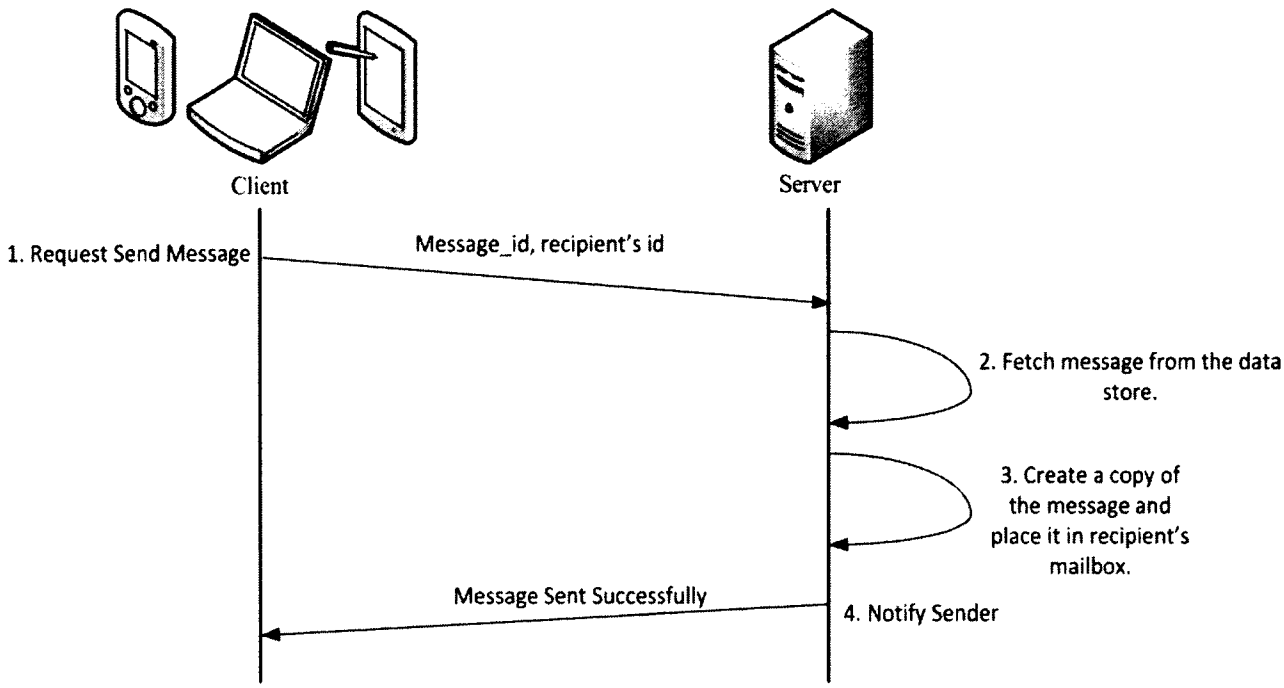


Figure 3.4.5 Forwarding an Existing Message

3.4.1.5 Terminating the Session

After accessing the data, the user must terminate the session. This prevents unauthorized personnel from accessing data while the user is away. The session can be terminated by simply sending a termination-message to the server. In addition to manual termination of the session, we recommend that organizations should enforce time-out policies which would automatically terminate the session after a certain time-period. This protects from any security breach in case the user forgets to terminate the session while away from the mobile device. By default we set the time-out to be two minutes, that is, the user is automatically logged-out of the service after two minutes of idle status. We also recommend organizations to enforce authentication time-outs to prevent brute-force attacks. A user should only be given certain number of attempts to prove identity. If a user fails to identify, he/she must be locked out and must contact the system administrator.

3.4.2 Data Encryption

The data stored in Data Store is always encrypted. The Web API encrypts data using a private-key encryption algorithm, AES [56]. AES was chosen as a standard encryption algorithm by NIST in year 1991. It is a block cipher that uses 128-bit block size and up to 256-bit key size. The API encrypts the data before storing it inside the Data Store, and decrypts it when data access is requested. Instead of encrypting/decrypting the entire database, SDSA individually encrypts all messages. Hence, only the requested message needs to be decrypted instead of the entire database. This improves the overall data read/write performance. The encryption/decryption key is stored at the server, and is accessible only by the Web API. Encrypting data inside Data Store protects data privacy from any attacks on the server, and prevents employees at server side without proper authorization to access sensitive data.

3.5 Contribution and Comparison of SDSA

The main contribution of our work is a low-cost, secure data sharing architecture that is customized especially for healthcare organizations. It enables safe exchanging of medical data over unsecure wired or wireless networks using mobile devices. The architecture can be integrated with any EMR application inside a healthcare system and provide extensions that facilitate data sharing over mobile devices. Unlike other data sharing solutions, our architecture makes use of HL7 specifications for data exchange ensuring maximum interoperability between EMR systems and mobile devices from different vendors. Our solution provides end to end data protection, including the time when data is on mobile devices, without any significant performance overhead. By moving most of the cryptographic computations on the server side instead of mobile devices, we have been able

to achieve significant improvement in time taken to send and receive data, versus other approaches where computations are mostly done on the mobile device. We present these results in next chapter.

The security protocol provides authentication for both - device and user, ensuring that data is accessed by authorized personal over an authorized device only. We use username-password based user authentication, and a custom authentication technique for mobile devices, in which every device is issued a cryptographically random API key. Since the API key is divided into two parts, with one part stored on the mobile device and other to be remembered by the user, we are able to protect the API key even when the mobile device is stolen. This solves one of the main problems with using certificates based device authentication. The certificates stored on the mobile device can be stolen and forged to be used with another unauthorized device. With our protocol, even if a part of key is stolen somehow, the information is useless without the second part of the key (a four digit number) that is known only to the authorized owner of the mobile device. By allowing only the Web API to access the Data Store with encrypted data, we ensure that sensitive data is not visible to unauthorized employees. Furthermore, our architecture is an open system which supports integration with various open source and proprietary technologies, unlike other commercial data security solution such as Blackberry Enterprise Server (BES) which is strictly proprietary and closed in terms of supported applications. Our architecture provides data security without requiring data to be transmitted to a third party Network Operation Center (NOC), which raises many privacy concerns such as data being visible by employees at NOC, and discourages many enterprises from using such solutions. Table 3.1 summarizes some important differences between our work and other existing data sharing solutions discussed in section 2.3.2.

	SDSA	**Other Medical Data Sharing Architectures**
Perform Device Authentication	Yes	No, only Transient Authentication performs device authentication.
Perform User Authentication	Yes	Yes
Integration with EMR applications and Lab Delivery Networks	Yes	No
Resources Required to Protect Data at Server	Low	High
Resource Requirements to Protect Data on Mobile Device	Low	High
Overall Complexity and Computational Cost	Low	High
Scalability	High	Low
Store Encryption/Decryption Keys on device	No, only a part of API key is stored.	Yes
Perform Resource Intensive Computations on Device	No	Yes
Require Additional Hardware/Software for data Security	No	Yes. Transient authentication requires key-fobs. Concord requires an additional mobile device to store keys.
Deployment Cost	Low	High, due to additional hardware/software required.

Table 3.1 Differences between SDSA and Other Data Sharing Architectures

3.5.1 SDSA versus Blackberry Enterprise Server

The Blackberry Enterprise Server (BES) is a commercial solution designed to extend an organization's communication methods to Blackberry mobile devices. It consists of various products and components that allow accessing confidential data such as mails, tasks, and calendars, over any Blackberry device that is registered with the organization. BES uses symmetric key cryptography to encrypt data during transmission over network, and while the data is stored on the device itself. It uses username-password based authentication, smart card based authentication, or combination of both to provide user authentication. Some important distinguishing features between our solution and BES are as follows:

1. **Difference in techniques used for device authentication**: By default, BES secures data by using user authentication only. Although, it offers an additional policy that enforces devices to authenticate itself before any data can be accessed. In order to perform device authentication BES uses a unique alpha-numeric PIN, which is assigned to all Blackberry devices by default for identification purposes. BES maintains a list of devices that are allowed to access data. The list is stored on the server located inside the organization. On every data access request, BES checks if the device's PIN is on the list of authorized devices. If the device is not on the list, data access is denied. There are two main problems with such an authentication technique. Firstly, the PIN is always stored on the device, and it can be revealed if the device is lost or stolen. An attacker may even forge an unauthorized device's PIN to be the same as the one stolen from an authorized device. This could be a critical security breach that could potentially allow an unauthorized device to be able to access organization's confidential data. Since the unauthorized device is not tied up to follow any

security policies which were enforced on the authorized device, BES might not even be able to revoke permissions, or delete data that has been already downloaded on such a device. Secondly, if an invader attacks the server, and is able to retrieve the list of authorized devices, same security breach would be introduced. Currently, the solution to prevent such an attack is by encrypting the list on the server. However, the PIN is still visible on the mobile device, and the first problem still applies.

In our architecture, devices are authenticated using cryptographically random API key that is used as device identifier. We compute a hash on this key using HMAC-SHA-256, and store the hash on server for device authentication. Even if the list consisting of device's hash is somehow revealed to an attacker, the original API key can still not be reproduced due to the irreversible nature of hashing algorithm used. Furthermore, we only store a part of the API key on the device which is useless without combining it with the other part, that is, the four digit passcode remembered by the user. Hence, even if the part stored on the device is stolen or revealed, our architecture still prevents security risks associated with BES' device authentication.

2. **BES forwards data to RIM's Network Operation Center (NOC)**: In order to exchange data within an organization, BES connects with organization's messaging server, its application server(s), and Research in Motion's NOC. To ensure that only legit data-access requests are received by BES, it accepts incoming transactions from only one place - NOC. When data needs to be sent from a Blackberry device to the server, it is passed to NOC which forwards it to the organization's server. This means RIM tracks all the transactions between the

device and the BES, which could discourage organizations dealing with highly confidential data from using this service. SDSA uses SSL to provide the same functionality, without sending data to any third party server that is not controlled by the organization. During the SSL Handshake protocol, the device and the server exchange important information for authentication. SSL uses a reliable transport layer protocol such as TCP to ensure that messages are being sent to intended recipient from a legit sender.

3. **BES is strictly platform dependent**: BES requires specific hardware and software configurations before it can be deployed. Minimum hardware requirements of BES are high computing multicore CPUs such as Intel Xeon processors, and at least three gigabytes of RAM. For an optimized performance, it is suggested that BES should be deployed on Windows server operating system only. These specific requirements, along with an expensive licensing fee to use BES, make it a very expensive solution especially for smaller organizations. BES works with Blackberry devices only, which means, organizations have to restrict its users to use these devices only.
4. **BES supports limited applications and lacks support for connectivity with EMR applications**: The types of data that can be accessed by using services provided by BES includes emails, calendars, tasks, notes, and files & documents. Basically, BES makes desktop applications such as Microsoft Outlook, or Lotus Notes available over Blackberry devices. In order to connect with the existing EMRs, any system must support standards specific to health industry, such as HL7 for exchanging medical data. Currently, BES doesn't use any such specifications for data sharing, and hence fails to directly integrate with the EMR applications. Since SDSA is customized specifically to healthcare industry, we

ensure all healthcare standards are followed, ensuring smooth integration with the existing EMR applications.

5. **BES stores data and encryption keys on the device**: BES allows storing encrypted data on the device. It uses eight different keys to encrypt/decrypt data, and encrypt/decrypt the encryption-keys itself. The encryption keys used for encrypting data are encrypted themselves by using two types of keys called ephemeral key and device transport key. Since keys are always stored on the device, including the master keys that encrypt other keys, the data could be revealed in case the device is stolen or lost. An attacker can find the master key stored on the device, and use it to decrypt all other keys and the data. SDSA on the other hand, doesn't allow storing any sensitive data on the device. Only a part of the API key is stored on device, which is useless to an attacker without the second part of the key.
6. **Difference in verifying message integrity:** BES uses symmetric key cryptography to verify data integrity. Every device registered with BES is given a private key, called as message key. Only BES and the device know the value of this key. The mobile device uses this key to encrypt data while sending it to BES over network. BES recognizes the format of a decrypted and decompressed message. A message is automatically rejected if it is in not encrypted with keys that BES recognize as valid [57]. A big drawback with this method is that it requires special keys to be stored on the mobile device and perform additional encryption operations in order to ensure data integrity. This introduces a performance overhead, and if the device is lost the encryption keys might be revealed. SDSA minimizes the amount of cryptographic secrets that need to be stored on the mobile device, and minimizes any computationally intensive tasks

such as encryption/decryption by moving it onto the server instead. This ensures that even if the device is stolen, minimal information is available for an unauthorized personal to access sensitive data. In order to provide message integrity services, SDSA use SSL protocol which doesn't requires storing any secrets/keys on the device.

3.5.2 SDSA versus Apple's iCloud

Apple's iCloud is an 'online-storage' solution that allows users to upload and store their personal data such as mails, photos, and music on a cloud (server) controlled by Apple. Users who potentially have multiple devices can benefit greatly from iCloud, as it provides them a central storage space. The best feature of iCloud is its capability to automatically synchronize data among different Apple devices owned by the user. It is still in beta testing, and the final release is due this year [58]. Even though iCloud allows secure access to data, it cannot be used in healthcare application. Following are some limitations of iCloud, and its differences from SDSA.

1. **iCloud does not support data sharing**: iCloud provides functionality that allow users to download data over multiple devices owned by the user. However, the data cannot be shared with other users. With healthcare applications, data sharing ability is a key requirement.
2. **iCloud lacks device authentication**: In order to access data stored on iCloud server, user needs to provide Apple id along with the password for authentication. Users could use any Apple device, and use their credentials to gain data-access permissions. This might be unacceptable in an organizational environment which requires only authorized devices to be able to access sensitive

data. SDSA provides a cryptographically strong device authentication technique to fulfill this requirement.

3. **The user authentication technique used by iCloud cannot be integrated with organization's current user management software such as Active Directory**: A user must have an Apple id to be able to access data. Since only Apple knows user ids, they cannot be shared with the organization. This means every user must have separate credentials for using iCloud services, and for using other legacy services hosted inside an organization. SDSA makes use of user's existing username-password registered with any user management service, such as Active Directory, for user authentication. This allows SDSA to successfully integrate within an organization's existing IT infrastructure.
4. **iCloud is only for Apple devices**: Like most commercial solutions, iCloud is strictly closed in terms of which devices can be used. It supports only Apple devices using iOS5 operating system. SDSA on the other hand, makes use of HL7 specification for sharing data on mobile devices with different platforms.
5. **iCloud server is controlled by Apple**: A major drawback that would discourage many organizations from allowing its users to store sensitive data on iCloud is that the data is stored on a server over which the organization has no control. This means organizations would have to depend on Apple to provide security measures or any important updates that are critical to data security. There is also a possibility that the employees at Apple who are otherwise unauthorized to view data, may be able to access data and reveal its contents. With SDSA, data is always encrypted when on the server, and only Web API can decrypt the data and provide necessary data access permissions. Hence, even if organizations chose to host its server outside its premises, the data is assured to be protected.

Chapter 4

Experimentation and Results

In this chapter we present design and implementation of the prototype built for testing our secure data sharing architecture proposed in chapter three. We discuss various test cases and outputs of rigorous testing performed on the prototype. We also evaluate security strength of our solution, and analyze how it protects data against various threats.

4.1 Introduction

To demonstrate our security architecture proposed in chapter three, a fully functional Point-of-Care-Testing (POCT) system was developed in collaboration with the team at Business Intelligence Research Group (BIRG), UNBC. POCT refers to medical testing performed at or near the site of patient care [59]. The motivation behind POCT is to reduce the time taken in conducting tests, and quickly provide healthcare providers with lab results, in order to assist them in making immediate decisions on patients' health. POCT includes blood glucose testing, hemoglobin diagnostics, cholesterol screening, and other tests which can be easily performed by using portable instruments that provide test results instantly. The main benefit of such tests is seen when the test output is immediately made available as an electronic record. Electronic records allow sharing results with other members of the medical team for their expert opinion. For this research, POCT is accomplished by using

mobile devices that communicate with our backend services running at the server, in order to collect patients' test data, or retrieve patients' existing medical information stored in EMR applications. We have implemented all components of SDSA, as discussed in previous chapter, and integrated them together to build an automated mobile lab delivery system.

In order to determine the usability and practicality of this research, we have closely worked with the physicians and technical systems analysts at Northern Health Authority (NHA), Prince George. Northern Health is one of the largest publically-funded healthcare providers that cater 300,000 people over an area of 600,000 square kilometers in the province of British Columbia [60]. With the valuable feedback from Northern Health, we have been able to refine our prototype such that it can be integrated with Northern Health's existing EMR applications and lab delivery network. This integration would allow physicians at NHA to use their mobile devices that are registered with NHA, to securely access patients' data stored in EMR applications at their clinic. Physicians would be able to use mobile devices while they are on field visits, to collect observations on patients' health. This information is saved electronically, and is transmitted to the Data Store located at the server for later use.

4.2 Prototype Design

We follow Service Oriented Architecture (SOA) to design our prototype. SOA is based on request/reply paradigm that separates an application's business logic from the presentation layer, and presents it as a set of services to client applications. Service is a term that refers to a well-defined function that can access data, perform various operations on it, and return the result to the service requester. SOA consists of three components – the service provider

which is the backend application located at the server side, the service consumer which is the front end application that directly interacts with users, and the service directory which contains description for all the services that are offered by service provider. The main benefit of using SOA is its platform independent nature. SOA components are loosely coupled and the service interface is independent of implementation technology. This implies that a service-provider can be implemented by using any language or platform, and does not necessarily need to be the same as the one used by service-consumer. This allows application developers or system integrators to build applications by composing one or more services without knowing the services' underlying implementations [61].

In our prototype, the service provider constitutes the Web API that integrates with the Data Store, and the EMR data transfer agent. It provides the necessary services for authentication and secure data exchange between the devices and EMR applications. The service consumer includes the client application that runs on mobile devices, and the EMR applications. Figure 4.1 shows different components of SOA.

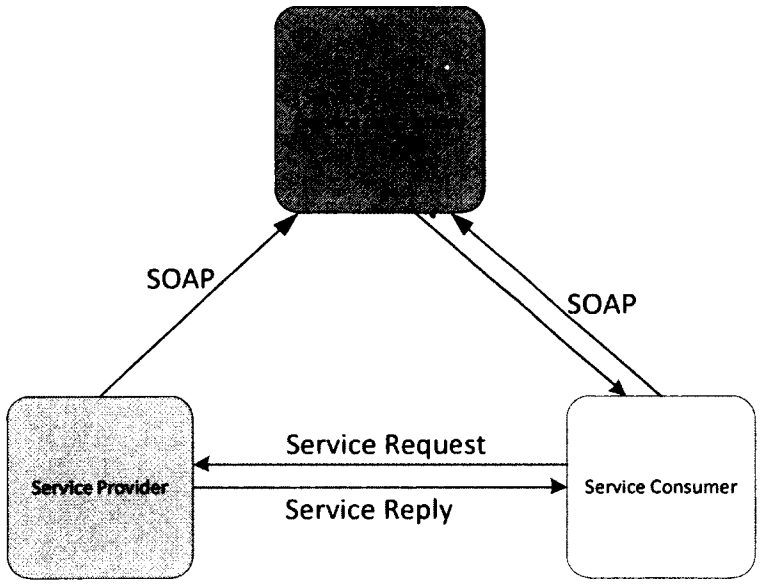


Figure 4.1 SOA Components

When the service consumer needs to access any services or resources, request is sent to the service provider over the Internet. The service provider and consumer follow Simple Object Access Protocol (SOAP) for message exchange. SOAP is a standard lightweight protocol recommended by World Wide Web Consortium (W3C) for exchanging structured information in a distributed environment [62] [63]. It is based on XML, and is platform & language independent. It is especially designed for communication between applications over HTTP and HTTPS, which is a better way of communication since it is supported by most of the Internet browsers [62]. SOAP defines a specific message format in which the XML messages are divided into two parts – the body and the header. The message body contains the actual message, and the header contains application specific information such as authentication, and payment. Figure 4.2 shows the mapping of SDSA's components to SOA.

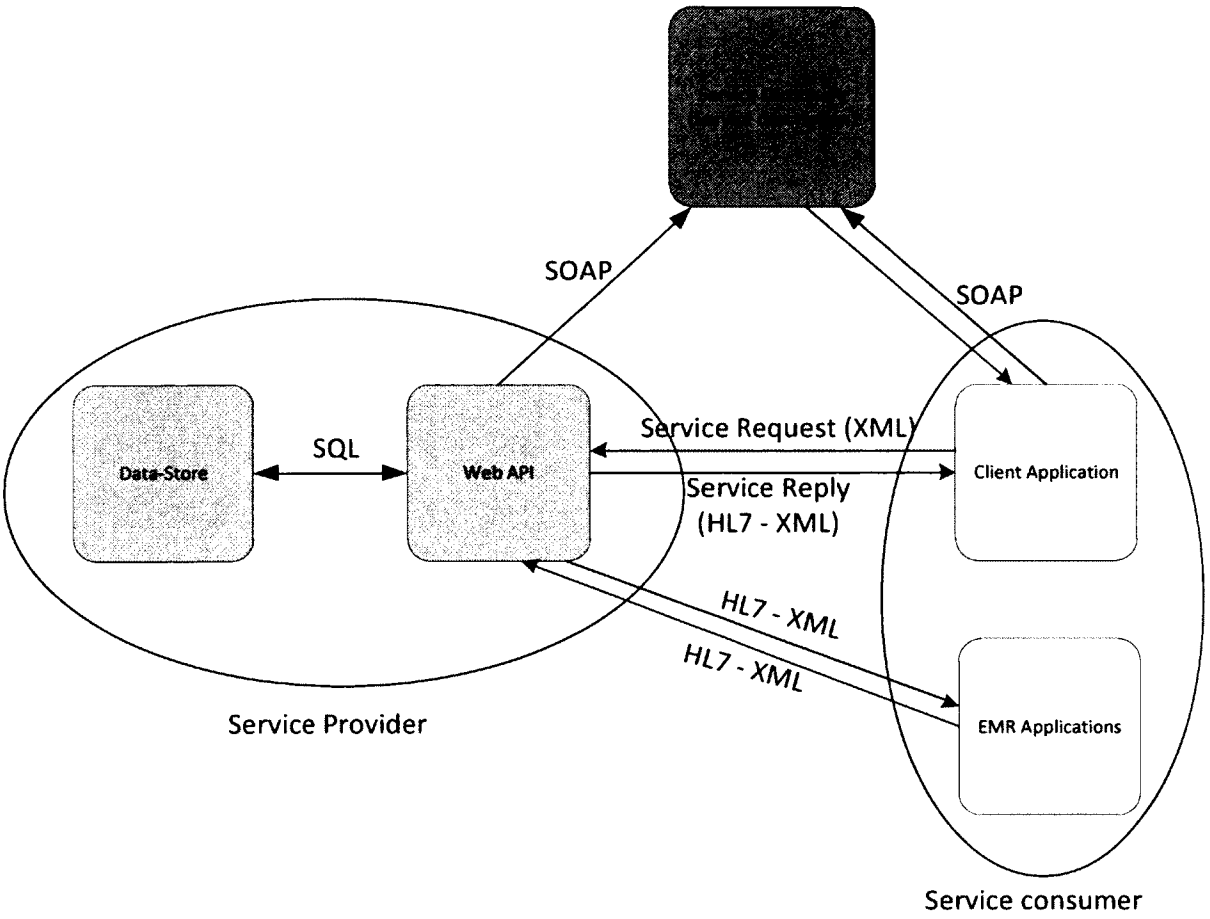


Figure 4.2 Mapping between SDSA and SOA

4.3 Prototype Implementation

As discussed in chapter three, SDSA consist of five components –Web API, Data Store, EMR data transfer agent, client applications, and an EMR application. The implementation details of all the components are described in the following sub-sections.

4.3.1 Web Application Programming Interface (API)

The Web API has been built as an ASP.NET 3.5 web application to provide various data exchange services to client applications. It uses C# as the underlying programming language, and directly communicates with the Data Store and the EMR data transfer agent. It uses default SQL Server 2008 connection manager provided by .Net framework, in order to communicate with the Data Store. All the methods defined in the API have been written as web methods, which allow us to create a standard service directory by using Web Service Description Language (WSDL). WSDL is an XML based language for describing the definitions of web services offered by the API, and methods to access them. The service directory is especially useful when different implementation technologies and platforms need to be used among different components of SOA. Table 4.1 on the next page shows various services along with their descriptions that are offered by the API.

Service Name	Description
FetchInboxMessageIds	Fetches the list of messages currently available in the user's inbox.
AcknowledgeMessageId	Marks message as acknowledged.
FetchDraftMessageIds	Fetches the list of messages currently available in the user's draft messages folder
FetchSentMessageIds	Fetches the list of messages currently available in the user's sent messages folder.
FetchMessage	Fetches a specific message from the database.
SaveMessage	Saves a draft message in the DB.
UpdateDraftMessage	Update the contents of a previously saved draft message.
SendMessage	Sends a message to mirth and the storage DB.
ForwardToEmr	Forwards a message to user's EMR application.
Authenticate	Authenticates a user against the Active Directory.
GetMessageId	Queries the DB for the MSP of a user based on the username.
AddMobileDevice	Registers a mobile device's information in the DB.
SearchForUser	Searches for existing users in DB, based on MSPs, first name, and last name.
GetUserAddressBook	Fetches the address book for a specific user in XML format.
AddToAddressBook	Adds a user to the address book.
MarkMessageDeleted	Marks a message as deleted.
UnmarkMessageDeleted	Removes a delete flag from a message.
FetchZdr	Extracts the ZDR segment from the HL7 message.
FetchProviderInfo	Extracts the provider's information from the HL7 message.

Table 4.1 Web Services Provided by the API

The SDSA Web API accepts SOAP based XML requests from client applications, and sends back the service results using the same. Any message/request that contains medical data is

formatted using HL7 specifications. For converting messages into HL7 specifications, the API uses a third party, java based open source HL7 broker called mirth-connect [64]. We select mirth-connect due to its widespread use in northern British Columbia, and ability to handle scalable message volumes at low licensing cost. Another option considered for the HL7 broker was Microsoft's BizTalk [65], but the proprietary nature and licensing cost made it unfeasible for this research.

Following the best practices for writing secure code, we encrypt and store all the connection strings from the API to different components in Web.Config file [66]. This keeps sensitive information such as username-password for database connection confidential. We have parameterized all the SQL queries used by the API in order to protect our code from SQL injection attacks[3]. Furthermore, instead of simply using string variables in the API to temporarily store sensitive information, we make use of SecureString class provided by .Net library. SecureString is a special class that encrypts the text when being used, and deletes it from the computer memory when no longer needed [67]. This protects our code from memory based attacks where an invader tries to access sensitive data from the shared computer memory.

4.3.2 SDSA Data Store

The SDSA Data Store is a Microsoft SQL Server 2008 database that acts as the central repository for POCT data including users, address books, devices, and messages. User data is used to identify POCT users within the system. The Medical Service Plan (MSP) number is used as the main identifier. Address books contain the information which allows a user to

[3] SQL injection is a technique for exploiting any security vulnerability of an application at the database level, by embedding modified database queries with the user input. Vulnerabilities are present when the user input on the front-end is incorrectly filtered for string literal and escape characters embedded in SQL statements [73].

quickly find other users for the purpose of sending a message. Device information includes the unique device id, associated user, API key, and the salt-value. Messages are stored in XML format identified by the message ID with a separate lookup table associating users to messages. This database is encrypted by using AES-256-bit encryption, and accessed only via the API to provide proper security. Standard Windows-based authentication and access procedures for SQL Server 2008 are used for accessing the database. Figure 4.3 shows the database schema.

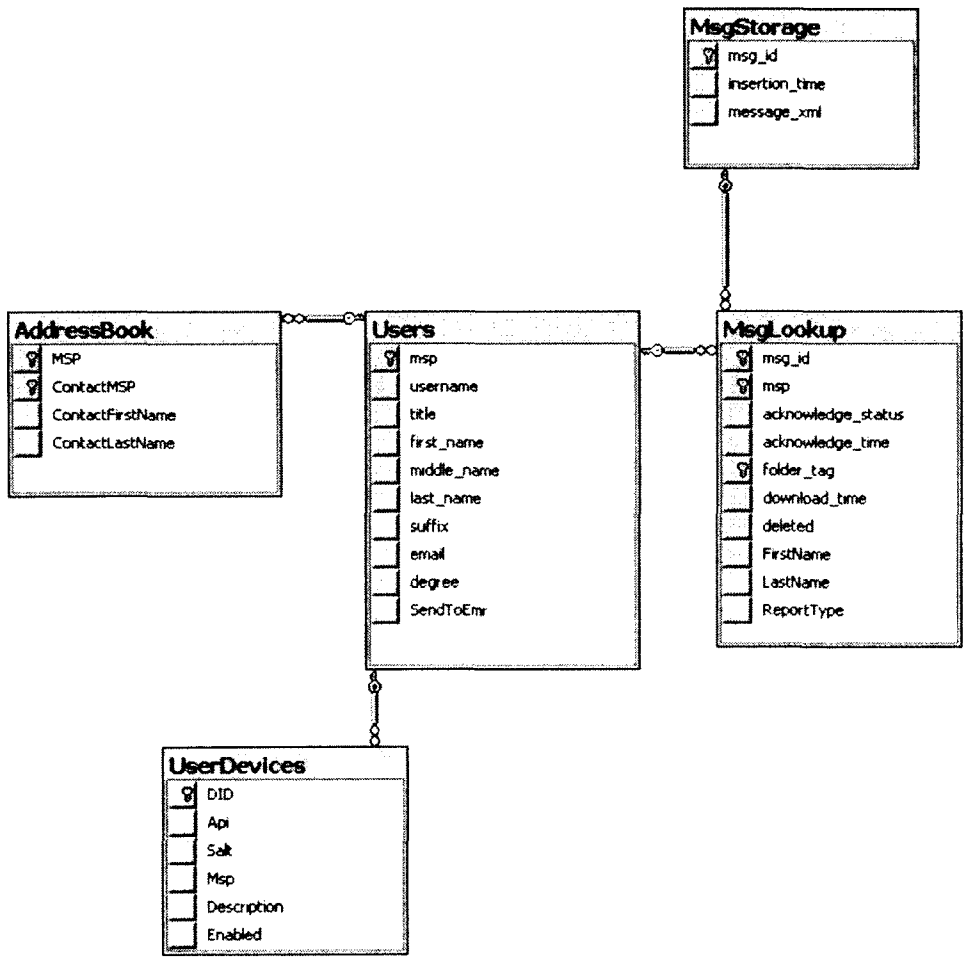


Figure 4.3 SDSA Data Store Schema

4.3.3 EMR Data Transfer Agent

The EMR data transfer Agent is installed on the machine that hosts EMR application. The agent periodically triggers a sequence of actions that allow it to download new messages from the EMR application and submit them to the API. It connects to the API and requests the timestamp of the last message submitted, and then uses it to get a list of message IDs from the API. These IDs are used to filter out duplicate messages. The EMR Data Transfer Agent queries the EMR database for the IDs of all messages with data and time greater than the timestamp received from the API. The message IDs are then checked against the list obtained from the EMR Data Transfer Agent and any duplicates are removed. The agent then iterates through the remaining message IDs and queries the database for all required fields for the HL7 message. These fields are assembled into an XML formatted HL7 message and transmitted over an authenticated HTTPS channel to the API Web Service.

4.3.4 Client Application

We have implemented an iOS based client application that can be installed on any Apple mobile device including iPad, iPhone, and iPod touch that runs on iOS 4.0 or above. The application is written in Objective-C[4]. It is built upon Model-View-Controller (MVC) design paradigm where the Model contains classes that encapsulate application data, View is made up of the windows, controls, and other elements that user can see and interacts with, and Controller mediates the logic between Model and View. The separation of responsibilities in MVC simplifies the design and maintenance of an application by allowing changes to different components independently. Figure 4.4 shows the MVC design paradigm.

[4] Objective-C is the native language supported by Apple devices.

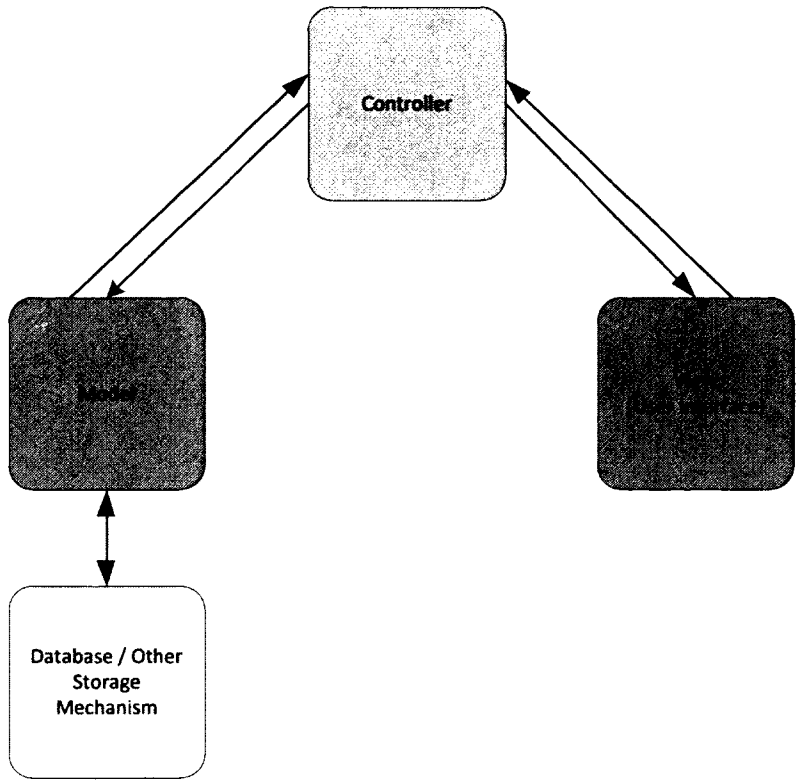


Figure 4.4 Model View Controller Design

Objective-C allows manual de-allocation and dereferencing of objects created throughout the application. This is better than automatic garbage collection especially when using devices with limited memory resources. Since automatic garbage collection can sometimes delay releasing resources, it is difficult to estimate and manage the actual size of the working set in memory. The delay also interferes with releasing other resources, such as descriptors and view handlers, which causes resource starvation. Garbage collection demands CPU time for memory management, but, with manual memory management this time can be saved. In order to ensure data confidentiality on the device, we restrict our

application to release all objects and dereference them to 'nil', as soon as our application is interrupted by another process running on the device, such as a phone call or receiving a text message. This protects data that is temporarily stored and being reviewed on device, from any malicious processes that could possibly manipulate device's shared memory to access the data. With manual memory management it is possible to restrict any unauthorized process or method call running on the device to access data. The user must establish a new session with the server every time an application is interrupted or intentionally closed. The only limitation with this is the inconvenience of re-establishing the session. Although, this could be improved if credentials can temporarily be cached on the device, as this would not require users to re-enter their credentials every time. Temporarily caching credentials presents many security risks and the solution to protect credentials would be different for different mobile platforms. A possible solution to this could be using hardware encryption, if supported by the mobile device, to encrypt the credentials while they are cached in the memory. However, securing the cache memory of the device is out of scope of our research work.

Since iOS Software Development Kit (SDK) does not include constructs to consume SOAP based web service, we use a third party open source tool, called SudzC [68], to generate the corresponding objective-C code. SudzC reads the application's service directory created in WSDL, and generates the corresponding source code that allows communication between the client application and the Web API. Current functionality of the application includes:

- Device and User Authentication: When the application launches on the device, users are prompted to enter their credentials (username-password and a four digit key). Every device securely stores a partial API key that was provided during device

registration. We use Apple's keychain[5] to store the key. On every data access request, user's credentials and the device's API key are sent to the Web API for authentication. The data access permission is given to the user on successful verification of user's credentials.

- Send Messages: Users can compose and send messages related to a patient's health. For example, a physician can send a message indicating lab results of a patient to another physician. The messages must include all the required fields of the CIX specification.
- View Messages: After successful authentication, the application downloads a list of sent/received/draft messages from the API and displays this list to the user; allowing them to view the content of each individual message. Users can also subscribe to receive a copy of messages that are stored in their EMR.
- Save/Compose Drafts: While composing a new message user can save the incomplete message as draft. This allows message composition to be resumed later. Using this feature, users can quickly take notes on patient's health during a home visit and complete the full message at a later time.
- Forward to EMR: A message composed on an iOS device can be forwarded to physicians' EMR to allow continued authoring of the message at a later date. This feature is dependent on support provided by the EMR application, and may not be compatible with all applications.

4.3.5 EMR Application

We use Medical Office Information System (MOIS) as EMR application due to its popularity and widespread use within northern British Columbia. MOIS is capable of integrating with lab delivery networks within healthcare organization, and receive lab results from various facilities inside the organization. It is primarily used by physicians for medical

[5] Keychain is a secure, encrypted container provided by Apple to store sensitive data such, as username-passwords, on the device. The keychain data for an application is stored outside the application's sandbox and is controlled directly by iOS. Only authenticated application processes are allowed to access application data stored in keychain [74].

office billing, scheduling, and documenting key elements of patient medical records including encounter notes, past procedures, prescriptions, allergies, consultation and referral reports [69]. We assume that all lab reports are delivered to MOIS, and it acts as data source for our prototype. The amount of patients' previous health information, such as previous lab reports, is limited by the amount of information stored in MOIS. Although, new data collected by using our client applications is independent of MOIS.

4.3.6 Error Handling and Logging

The implementation provides several levels of error handling and logging, offered at the Web API, the Data Store, and the client application. The Web API is written to catch any errors and exceptions in performing API service calls. Whenever an exception is raised, the API returns useful error messages indicating the reason for failure. The client application can use these error messages to debug the problem. Figure 4.5 shows the client application displaying an error message received from the API. The client application logs various actions performed by the application by using NSLog and NSAssert classes available in Objective-C. We also store a log of all the activities performed on the data-store by using SQL-Server's logging system. This information is particularly useful while debugging errors, and when administrator needs to trace and analyze various activities on the database.

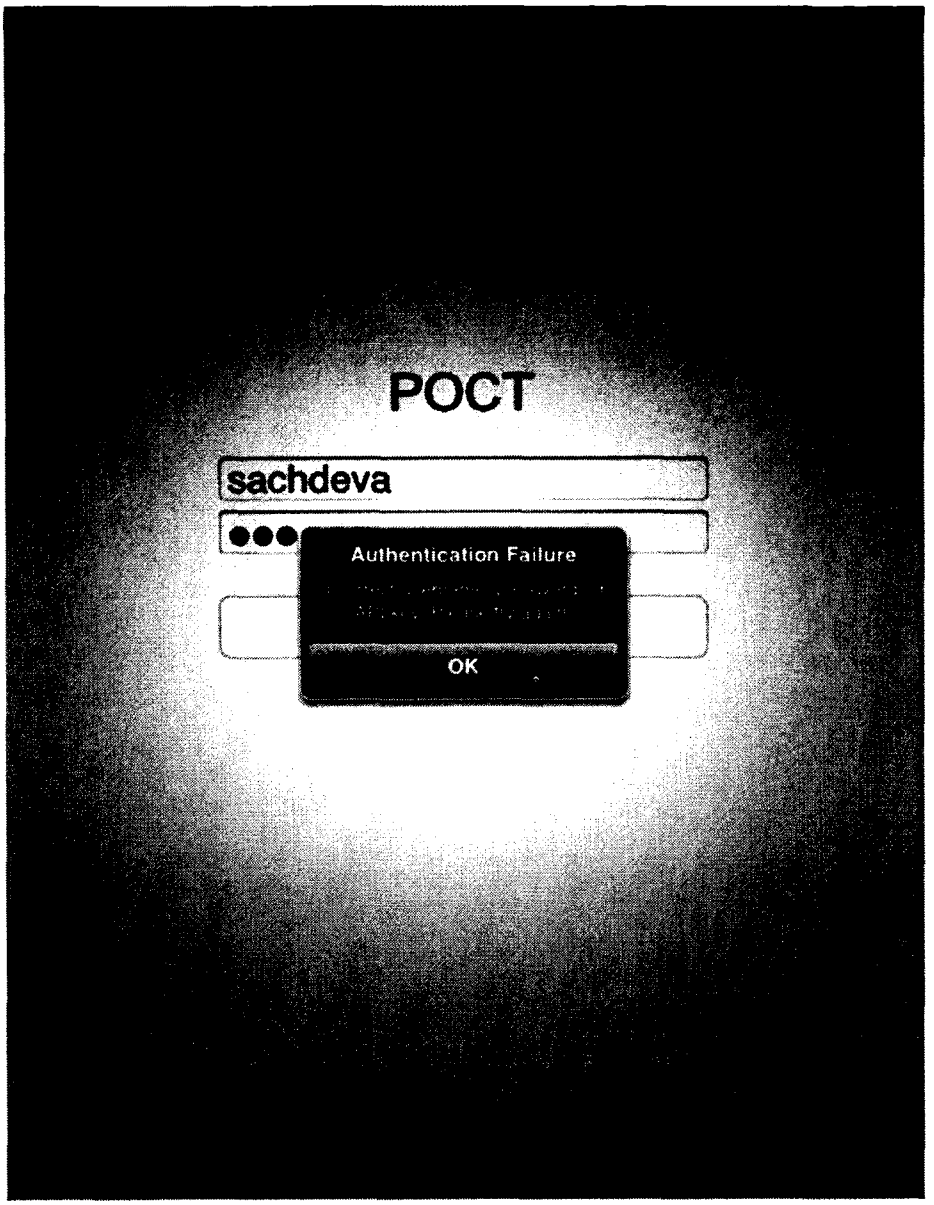


Figure 4.5 Error Message Received From the API

4.4 Experimentation Requirements

By simulating a mobile POCT system we have been able to perform series of experiments at BIRG research lab to determine the feasibility of our proposed architecture. Through these experiments we analyze the usability, deployment cost, security, and scalability of the prototype. We use standardized performance monitoring tools and applications, such as

Apple's Xcode performance instruments, to collect and store our test results. To obtain accurate results, the performance testing is done on the mobile devices, and on the server. We define usage scenarios for our prototype in section 4.4.1. The specific criteria that are used for evaluating the prototype are defined in section 4.5. We have divided the evaluation criteria into two broad categories – usability and technical. Usability criteria include requirements that should be fulfilled from user's perspective such as assessing user friendliness, platform independency, and ease of software installation. The technical criteria focus on requirements such as memory and CPU resources consumed by the application to perform various operations.

4.4.1 Usage Scenario

There are two main scenarios where our prototype can be used.

1. **Using POCT system inside a clinic or hospital**: Our solution can be used by healthcare providers, inside their clinics or hospitals. In this case, the POCT client application, installed on their registered mobile devices, is used to note observations on a patient's health, and to compose new EMRs. These EMRs can be forwarded to physician's EMR desktop application that directly connects with existing lab delivery networks inside a clinic or a hospital. The client application allows physicians to create quick drafts on the mobile device and store it in POCT Data Store. These drafts can be resumed later and can be forwarded to EMR application on completion. Using our system, physicians can also request to download lab results of a patient from the EMR application. These results, by default, are forwarded by various facilities within the hospital to the EMR desktop application. The user establishes a secure session with the Web API, which extracts the lab results from

EMR application and makes it available on mobile devices. Our client application has the capability to embed various templates that can be used to collect medical data of different types. Such templates include blood-glucose tests, clinical summaries, referrals, prescriptions, and general observations.

2. **Using POCT system during a field visit:** In this scenario, we assume the user is travelling and cannot access the local-network that is available from within the organization. An organization usually implements security mechanisms, such as authentication with Active Directory, to protect its services from unauthorized access. However, additional security measures must be applied when accessing these services from outside the local-network. Our prototype allows users to use all the services, as described in the first scenario, while roaming. By using special device and user authentication mechanisms, the access of services while roaming can be considered strongly authenticated. In order to access the services, the mobile traveler establishes a secure session with the Web API by using HTTPS connection over the Internet.

4.5 Validation against Criteria

4.5.1 Usability Criteria

Criterion 1: The solution should be easy to use.

The User Interface (UI) of client application needs to be simple and intuitive. Since mobile devices have limited screen size, the UI should not be congested for users to input patient data. It is important that the text is clearly displayed and has a readable font size. The client application should be responsive and should be free from delays in view transformations.

The application should also be easy to install for users and administrators. This criterion also contributes to commercial viability of the solution.

Validation: Our client application has a very simple navigation-based, default iOS application design. The application is carefully designed to be similar to iOS default mail application, so that it looks familiar to users. The application consists of clear text menus that are intuitive and easy to read. It requires no additional software to be installed on the mobile device, than the application itself, in order to connect with rest of our system. The application displays error messages by using pop-up notifications, in case of any exceptions related to either data entry or any other failure during message transmission. Figure 4.6.1 and 4.6.2 shows screenshots of the client application.

Since the Web API runs as a background web service on the server side, minimal interaction is needed by the administrator. For device registration, a C# console application has been built, that allows administrators to generate a cryptographically random device-API key. The administrator enters the information about mobile device and the end user by using command line input. All other components of SDSA run as background services which are configured to automatically start on system boot. Everything on the server side, except for device registration, is automated and requires minimal user-interaction. Hence, we consider criterion 1 to be fulfilled.

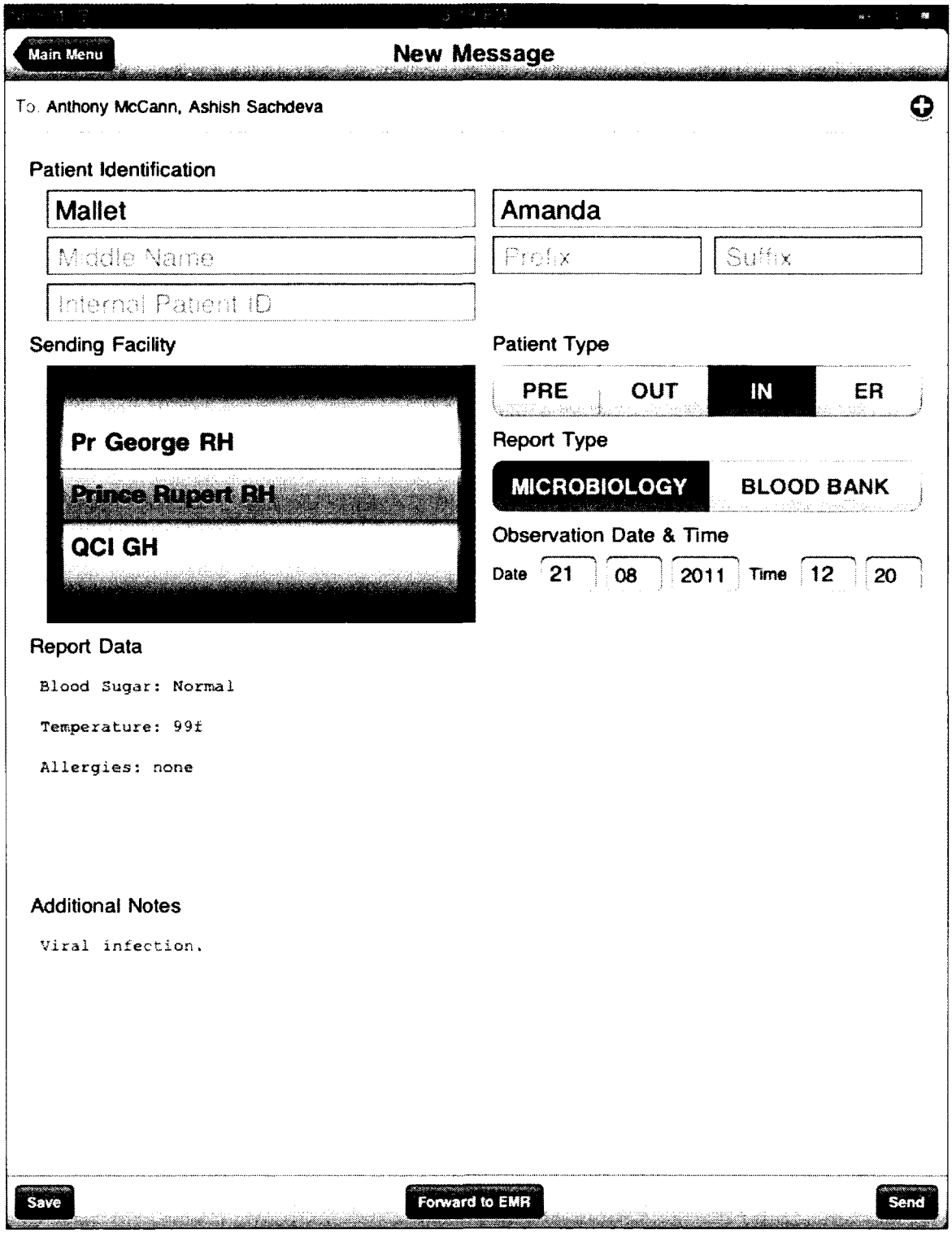


Figure 4.6.1 iOS Client Application Screenshot – Compose Message.

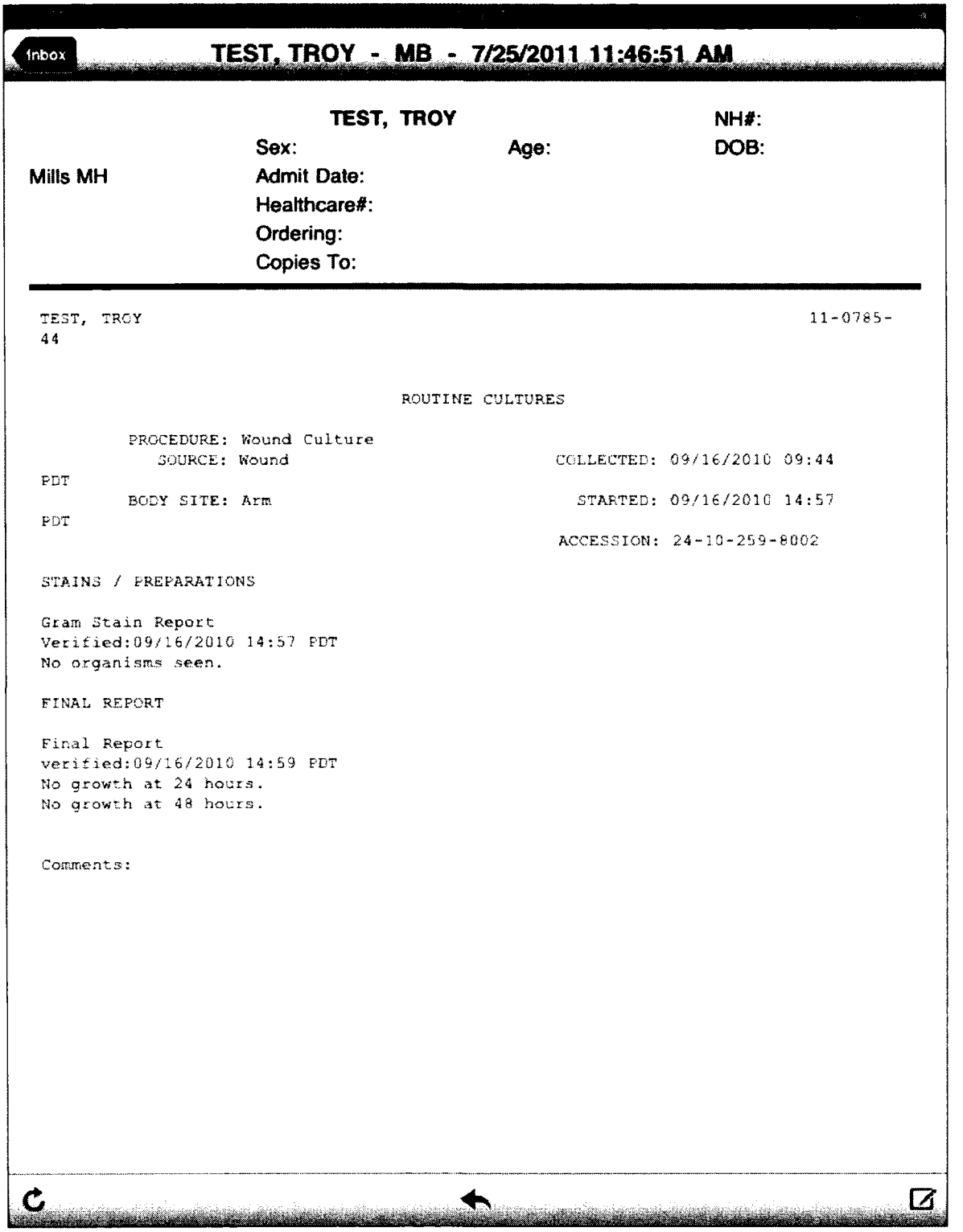


Figure 4.6.2 iOS Client Application Screenshot – View Message.

Criterion 2: The solution should be platform independent.

The solution should not be limited to a particular platform or a set of platforms.

Validation: We use ASP.Net framework to build the web service API. The functionality of web service is platform independent, and would remain the same even if it is written using any other programming language. We restrict the web service to accept XML formatted strings as input, and for sending results. This allows smooth integration with mobile devices of different platforms, as long as client applications can parse XML formatted strings. XML itself is a platform independent standard for data exchange. We use WSDL to create the service directory which is accessed by client applications written in different programming languages. Our Web services are currently hosted on the server using Internet Information Services (IIS) 7.0. However, the services are compatible with many other web servers as well. Hence, we consider criterion 2 to be fulfilled.

4.5.2 Technical Criteria

Criterion 3: The underlying cryptography algorithms must be NIST certified.

It is important to select strong cryptography algorithms for encryption and hashing purposes. These algorithms are the core of any security solution. The level of security provided by any solution directly relies upon the security strength of these algorithms. It is also important to use only those algorithms which are popular and are widely used in industry, to ensure maximum compatibility between different components of our architecture.

Validation: All cryptography algorithms used in our solution have been recognized as standard algorithms by NIST. We use HMAC-SHA1-256 algorithm as our hashing

algorithm, and AES-256 as encryption algorithm. These are cryptographically strong algorithms that are broadly used in industries dealing with highly sensitive data. Hence, criterion 3 should be considered fulfilled.

Criterion 4: The keys should be generated using a secure random number generator and must be of standard length as recommended by NIST.

All security keys must be generated in a secure way by an authorized entity only. The keys should be unique and random for every mobile device. It is important that a strong cryptography algorithm is used to generate random numbers, otherwise the security strength of the key is considered weak.

Validation: All keys used in SDSA are generated by using a specially designed C# console application. The application uses RNGCryptoServiceProvider class [70] provided by .Net cryptography library to generate random numbers. This is considered a standard technique to implement secure random number generator. As recommended by NIST, we use 256-bit key length for our hashing algorithm, and the same for encryption algorithm. Hence, criterion 4 should be considered fulfilled.

Criterion 5: The private keys should be securely stored on the device and server.

If the private keys are compromised it defeats the whole purpose of performing authentication of mobile user. It is important that these keys are stored securely and can only be accessed by authorized users/entities.

Validation: Storing keys on the device can cause critical security breach in any solution. If the keys are revealed, the data can be considered compromised. SDSA is designed to minimize the number of keys that need to be stored on the mobile device. Since, most of the

cryptography operations are moved to the server side, mobile devices store just a single API key used for device authentication. To protect the key while it is stored on the device, we divide it into two parts, and store only one of the parts on the device. Hence, even if the partial key stored on the device is compromised, it is useless without the other part that is known only to the authorized owner of the device. To further protect the key, we store the partial key in a secure storage space provided by mobile devices, such as Apple's Keychain. It is assumed that every mobile device that is supported by SDSA would already have a default built-in secure storage space that can only be accessed by authorized application and processes.

In SDSA, the server stores just a single key for encrypting/decrypting data in the data-store. This key is encrypted and stored in web.config file of the Web API. When a new device API key is generated for any new device, the server uses HMAC-SHA1-256 algorithm to calculate a hash on the key. The server stores this hash value for device authentication, and discards the original key after it is distributed to the user. Since hashing function is irreversible in nature, even if the hash value is revealed corresponding key cannot be recomputed. Thus, we consider SDSA secures keys while on the device or at the server. Hence, we consider criterion 5 to be fulfilled.

Criterion 6: In case of loss or theft of mobile device, data must remain protected from unauthorized access and must be able to recover on a new device.

If a mobile device is lost or stolen, the patient data must remain secure. Under no circumstances an unauthorized user should be able to access patient data, or reveal any other sensitive information (such as private keys) that would cause a security breach. It is

important that if a mobile device is damaged or lost, user should be allowed to restore all the data and application settings on the new device.

Validation: Unlike other data sharing solutions, SDSA enforces that data should be consolidated into a single repository stored at the server side. Data of every user is stored on the server, and not on the mobile device. Hence, even if the user loses the device, all the data can be restored on a new device. The user must request the administrator to register the new device with the Web API, by generating a new device API key and discarding the old one. Since, the old key is discarded the lost mobile device is deauthorized and cannot be used to access data anymore. Hence, we consider criterion 6 to be fulfilled.

Criterion 7: The sensitive data should remain protected and secure from employees at mobile telecommunication provider.

When data is exchanged between the mobile device and the web API, it must be protected from employees at the telecommunication providers. This is an important requirement, as the telecommunication providers can sometimes route data packets to locations which are outside the home country of the hospital. This can raise many privacy concerns, and would discourage organizations dealing with confidential data from using such solutions.

Validation: Some data exchange solutions, such as BES, requires data to be forwarded to a third party Network Operation Center (NOC). The data packets, exchanged between the server and the mobile device, remain at NOC until the recipient is ready to receive data. This introduces a critical security risk. SDSA does not forwards data packets to NOC. It instead establishes a direct SSL connection between the server and the mobile device, and performs mutual authentication. The data is divided into small packets which are encrypted individually. The data is sent over the network by using a reliable transport layer protocol,

such as TCP/IP. This ensures that the data remains protected during transmission and would be delivered to correct recipient only. We consider criterion 7 to be fulfilled.

Criterion 8: Data should be protected while it is stored at the server.

Organizations usually host servers in a virtualized environment, with same server providing different type of services to different user groups. This helps in optimizing system resources of the server, and conserves resource requirements of an organization. The Data stored at the server must remain secure from any personnel who is not authorized to access patients' data, but still has authorization to access other services hosted by the server.

Validation: The data stored in Data Store is always encrypted and cannot be accessed by any component of SDSA other than the Web API. Only API has access to the decryption key. In addition to storing data as encrypted, the data-store requires user authentication to provide access to administrators. The user authentication is linked with Active Directory, which categorizes users in different groups based on their credentials. In our implementation of SDSA, we have configured the Data Store, a SQL Server 2008 relational database, to be visible only to users who are classified under administrator user-group. The data inside remains encrypted to ensure data confidentiality. Hence, we consider criterion 8 to be fulfilled.

Criterion 9: Data must be protected while being viewed on the mobile device.

When the user establishes a session with the web API for accessing data, it is important that data remains secure throughout the session. Once the session is terminated, no trace of data should be available on the mobile devices. This is an important requirement to protect data from any malicious code attempting to access sensitive data.

Validation: When user establishes a session with the Web API, in order to access the data, data-access permission is granted by the server for a fixed time-interval (e.g. five minutes). The user must re-establish the session after the fixed time interval. If there is no activity from the user for about five minutes, our client application automatically terminates the session and locks out the user. The user must provide all the credentials again to reestablish the session. If the user continues to use the application beyond the fixed time interval, then instead of locking out the user, the application automatically extends the current time limit. When an application is closed by the user, or minimized, or interrupted by another application, all the objects created by the application are dereferenced and released. Once the application is closed, no data trace is available. Hence, we consider criterion 9 as fulfilled.

Criterion 10: Data read/write operations must be performed within an acceptable amount of time.

This is one of the most important criterions that evaluate the overall performance of any security solution. It is important that users are able to access data within a reasonable amount of time; otherwise it might discourage them from using the solution.

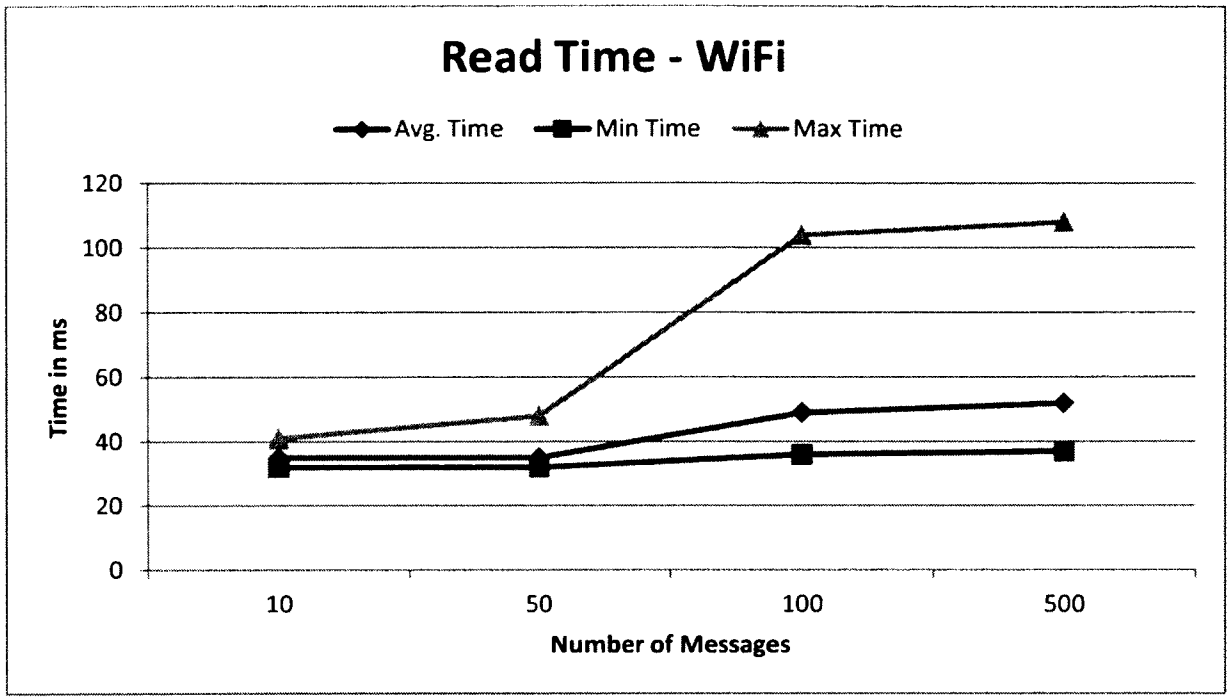
Validation: The time taken to perform read/write operation includes the time spent in performing user/device authentication, time taken to transmit data over the network, and time taken by the server to format the message with HL7 specifications. In order to monitor the performance of our system, we ran series of tests on a first generation iPad. We created various tests cases using different parameters, such as number of messages sent/received, message size, and connection type. The data was recorded by using Apple's Xcode performance instruments. The results of each test case are discussed as follows:

Read/Write performance by network connection

We have designed two test cases for recording the read/write time based on type of network connection. In first case, we run the tests using a high speed Wi-Fi network with various batches of messages. This case simulates the usage scenario where the user accesses data from within an organization. In second case we simulate mobile traveller by using a 3G network connection to send and receive messages. Four different batch sizes, ranging from 10 to 500 messages per batch were used for testing. The sample message contains textual information, and is 4KB in size. We record time taken to read and write individual messages, and calculate the average time taken by each batch. We also record the maximum and minimum time for each batch to represent the best case and worst case scenario. UNBC's Wi-Fi network and 3G connection by Bell, Canada was used to run the tests. Table 4.2 shows the network speed that was available for running the tests. Figure 4.7 and Figure 4.8 shows different graphs representing the results of our tests.

Connection Type	Downstream	Upstream	Latency
Wi-Fi	4.95 Mbps	.733 Mbps	45ms
3G	0.92 Mbps	0.33 Mbps	324ms

Table 4.2 Network Connection Speed



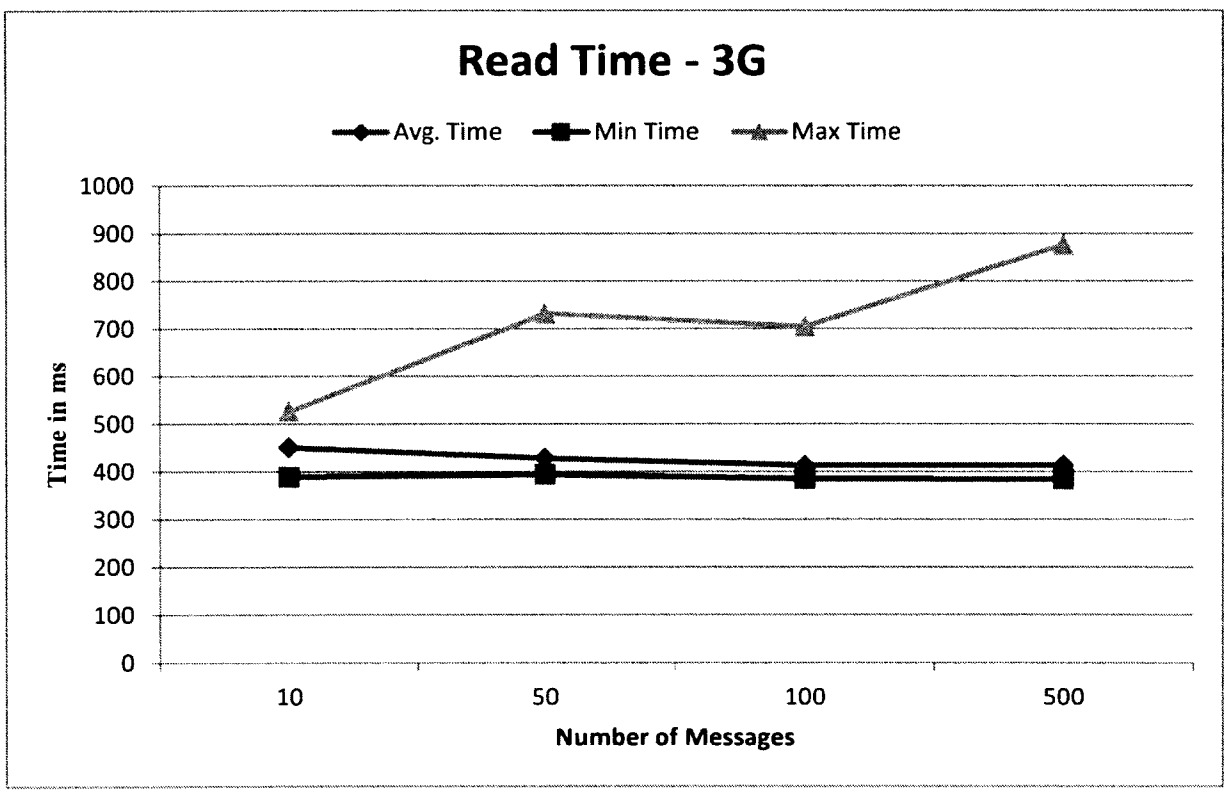
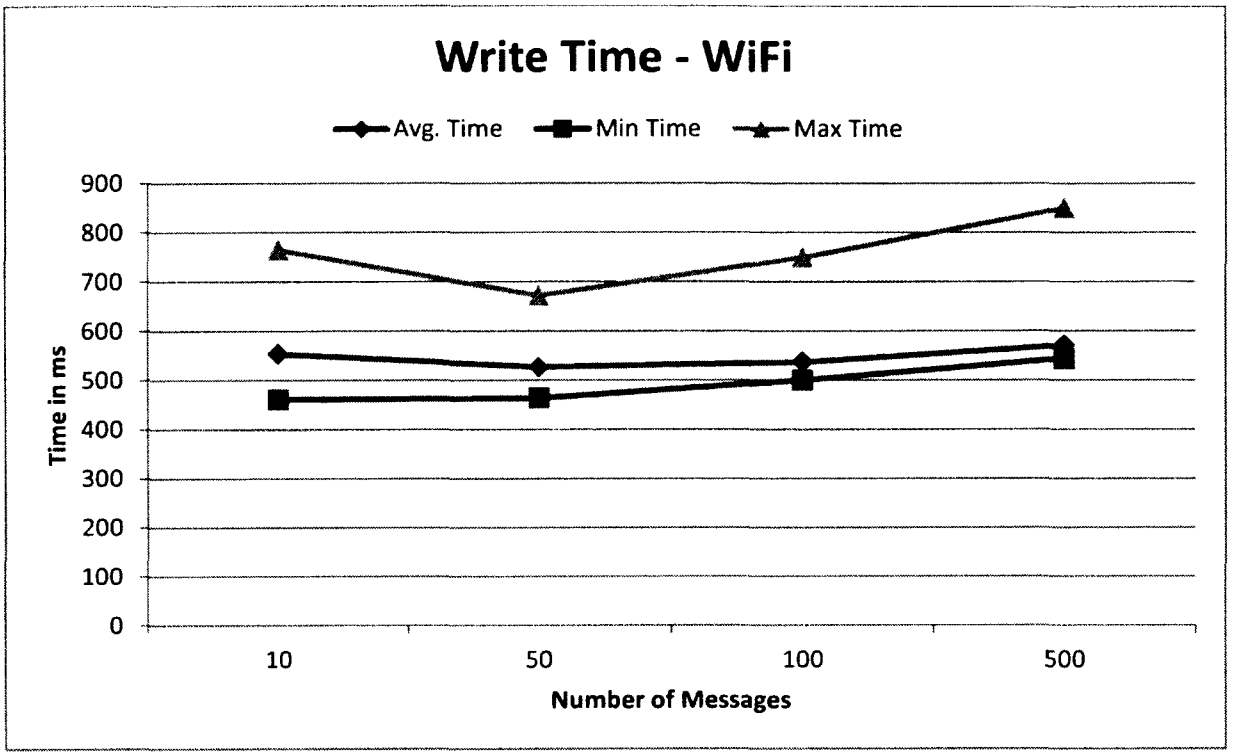


Figure 4.7 Read Time using SDSA



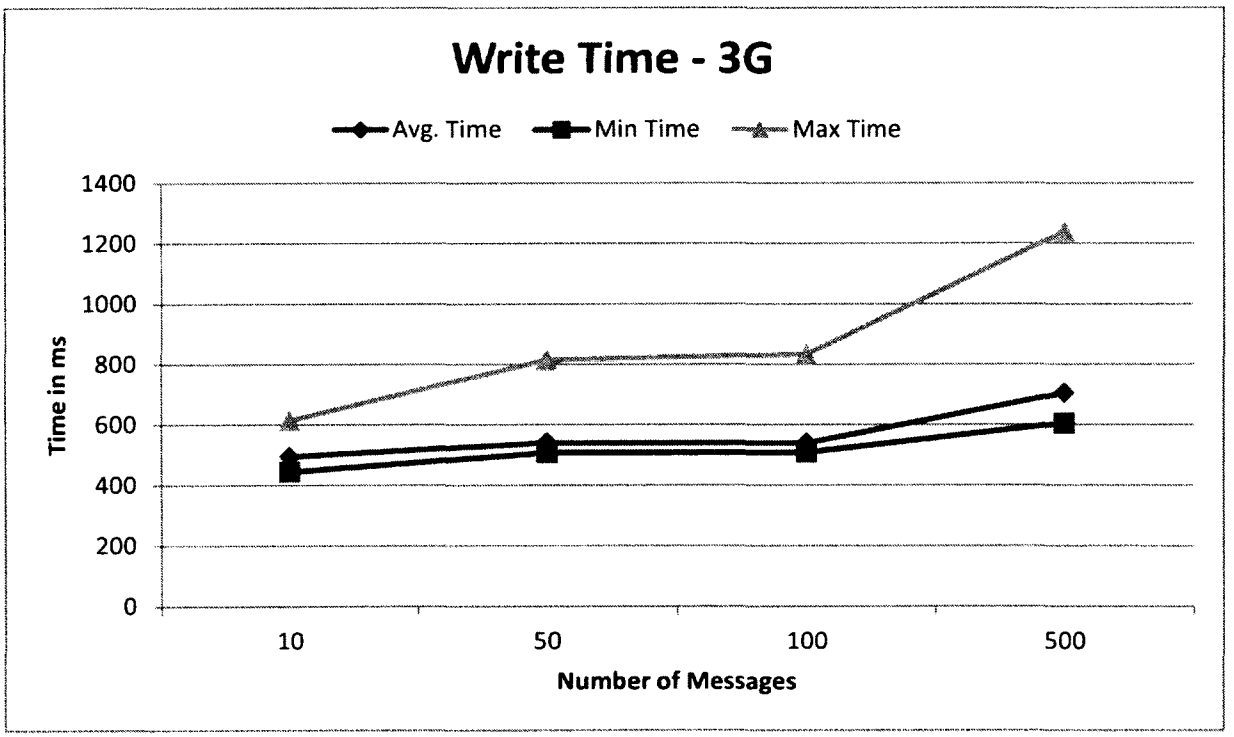


Figure 4.8 Write Time using SDSA

From the graphs shown in figure 4.10 and 4.11, we find that the average time taken to read or write messages significantly varies between 3G and Wi-Fi network. On an average, write time over Wi-Fi is 23ms (4.03 %) less than 3G. The read time is 383.5ms (89.97%) less than 3G. The improvement in message read/write time when using Wi-Fi is due to its faster downstream/upstream speed and low network latency of only 45ms. Network latency is dependent on many factors such as propagation delay, router delays, computer hardware delays, and interference due to weather and electromagnetic signals. It is one of the key factors that affects the network performance, and therefore affects the read and write times. Unfortunately, we cannot control high latency involved with the 3G networks. Eliminating network latency is a broad problem, and is out of scope of this work.

In our testing, we observed that while reading/writing messages in batch mode, there were few random peaks in read/write times caused due to network latency as well as the application latency. Application latency involves the delay in manipulating and presenting data, caused due to intense disk input/output operations and hardware limitations such as amount of main memory. This causes the average maximum time to be exceptionally large. Although, the average time taken for sending/receiving messages always remains close to the minimum range. This implies that most samples are unaffected by random latency peaks. Table 4.3 summarizes the average data read/write times.

	3G		**Wi-Fi**	
	Read	**Write**	**Read**	**Write**
Average Time [ms]	426.25	570.25	42.75	547.25
Average Min Time [ms]	388.00	516.75	34.25	492.25
Average Max Time [ms]	709.95	876.00	75.25	759.25

Table 4.3 Message Read/Write Times

Read/Write performance by message size

In order to evaluate the effect of message size on the performance, we calculate the average time taken by the iPad to send and receive sample messages of varying sizes. We ran five test cases that use different sample messages of size 4KB, 8KB, 10KB, 50KB, and 100KB. Each test case was run in a loop of hundred, and the average time taken to read and write a message was recorded. Figure 4.9 shows the results of our tests.

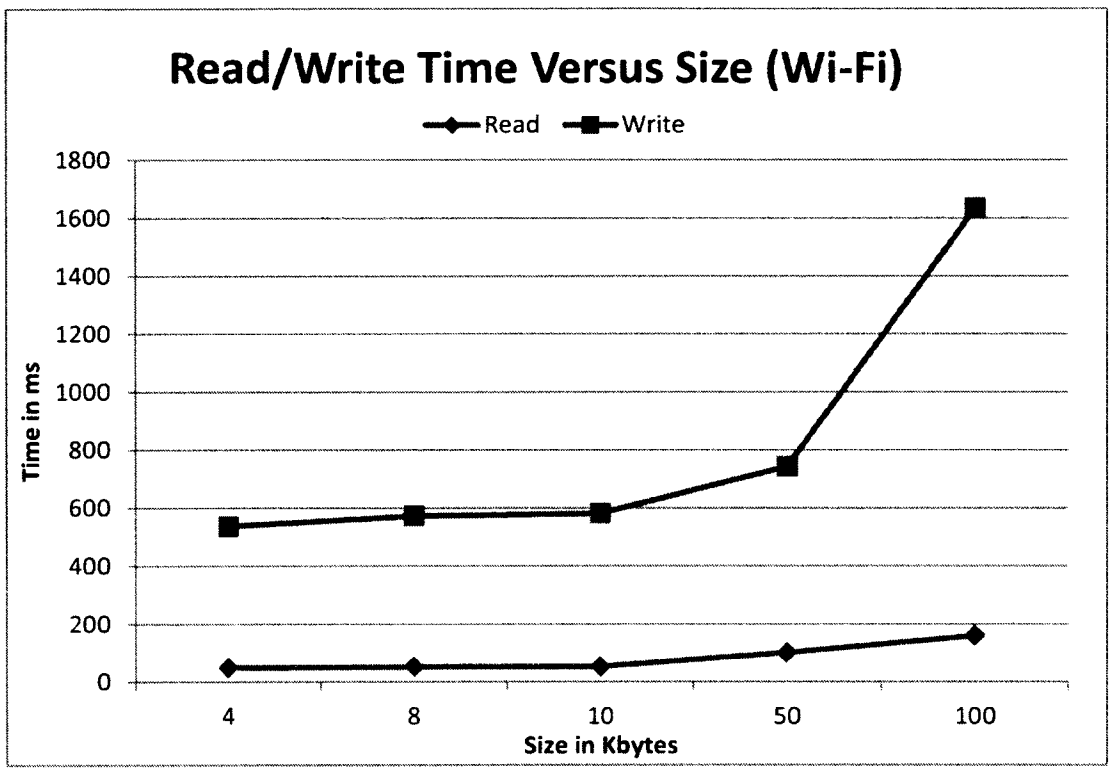


Figure 4.9 Read/Write Time versus Message Size

Comparison with other data sharing solutions

Due to unavailability of performance results of other data sharing solutions, we are limited to compare our test results with Secure Sharing Scheme (SSS) only. Furthermore, SSS is the

only mobile data security solution that is designed specifically for data sharing among group members. In order to perform fair comparison, we use Wi-Fi network connection with 2.4Mbps downlink and 153Kbps uplink speed to meet the specifications as published in [20]. Since SDSA performs most cryptography operations on the server instead of the device, it significantly outperforms SSS both in terms of time required to read and write data. Table 4.4 summarizes the comparison. We consider criterion 10 to be fulfilled.

	SDSA			SSS		
	10KB	50KB	100KB	10KB	50KB	100KB
Write	1	3.77	5.45	4.18	11.9	21.5
Read	.05	.09	.15	2.99	3.30	4.91

Table 4.4 SDSA versus SSS

Criterion 11: On-device resource consumption must be optimized.

Most security solutions are highly resource intensive. Due to limited resources available on mobile devices, it is important that the security solution must optimize resource utilization.

Validation: To determine the on-device resources used while using our application, we record CPU and memory usage on the mobile device using Xcode performance monitoring instruments. Figure 4.10 shows a snapshot of device's CPU activity, while the client application is being used. Our application only takes 13 MB of disk space on the device. Since most of the cryptographic operations are performed at the server, the only overhead involved at the client application is for establishing the SSL connection and sending/receiving SOAP requests. The CPU usage is 0-25% while browsing contents of the application. The application takes about 36.53 MB of physical memory. We consider our

application to be a low resource consuming application, and consider criterion 11 to be fulfilled.

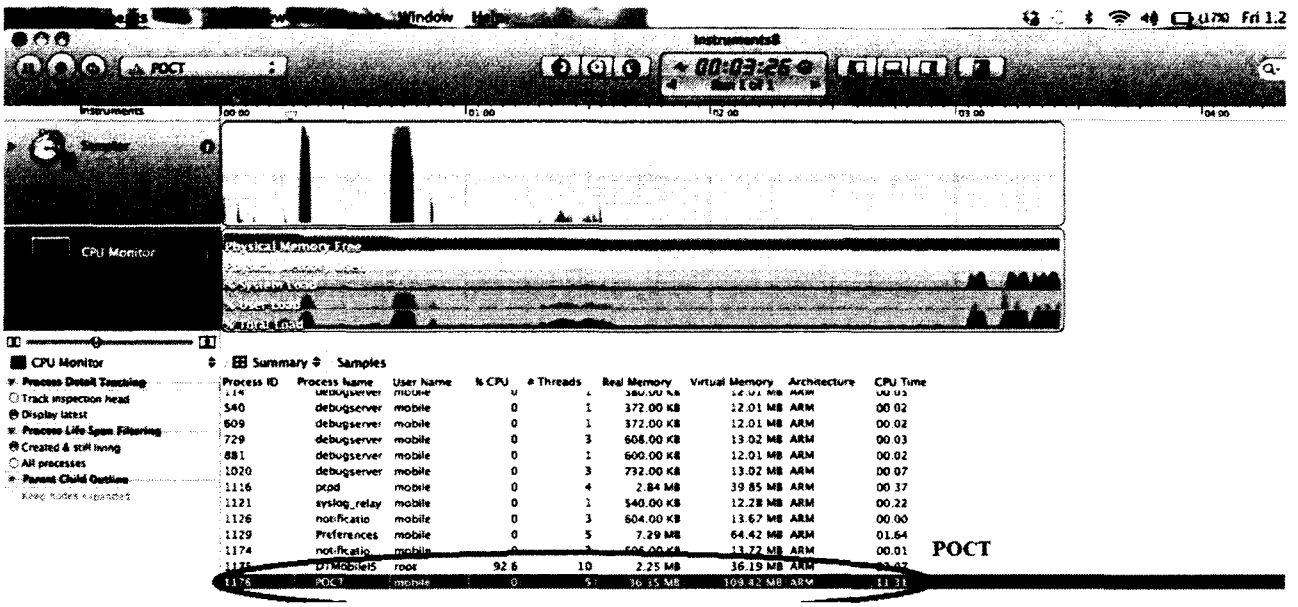


Figure 4.10 CPU Performance Monitor

Criterion 12: Solution must support both Wi-Fi and 3G network connections.

Mobile devices not always support Wi-Fi or 3G network connectivity. It is important that the solution must be able to provide its services over Internet regardless of the type of network connection used. This would allow using the solution while the user is travelling and does not have any Wi-Fi connectivity.

Validation: Our solution is compatible with both Wi-Fi and 3G network connection. We have tested our application with both connection types, as discussed in validation for criterion 10. Hence, we consider criterion 12 to be fulfilled.

Criterion 13: The solution must be scalable.

The solution must be scalable to accommodate growing number of users and mobile devices used by them. Users must be allowed to register more than one mobile device to be able to access the services provided by our POCT system.

Validation: SDSA allows multiple users to register for the services provided by the Web API. Users are allowed to register multiple devices without any restrictions. The Web API is hosted on a virtual machine running Windows Server 2008 platform. The virtual machine utilizes single core processor with speed up to 3.2 Ghz, and 4 gigabytes of memory. We use IIS 7.0 webserver to host our web services. In order to load test the functional behaviour and measure performance of our webserver, we use soapUI performance monitor [71]. Using the performance monitor we collect the average time taken to read/write a 4KB message, when multiple users request data-access concurrently. We created test cases with different number of users – 10, 50, 100, 250, 500, 750, and 1000. Figure 4.11 and Figure 4.12 shows the results.

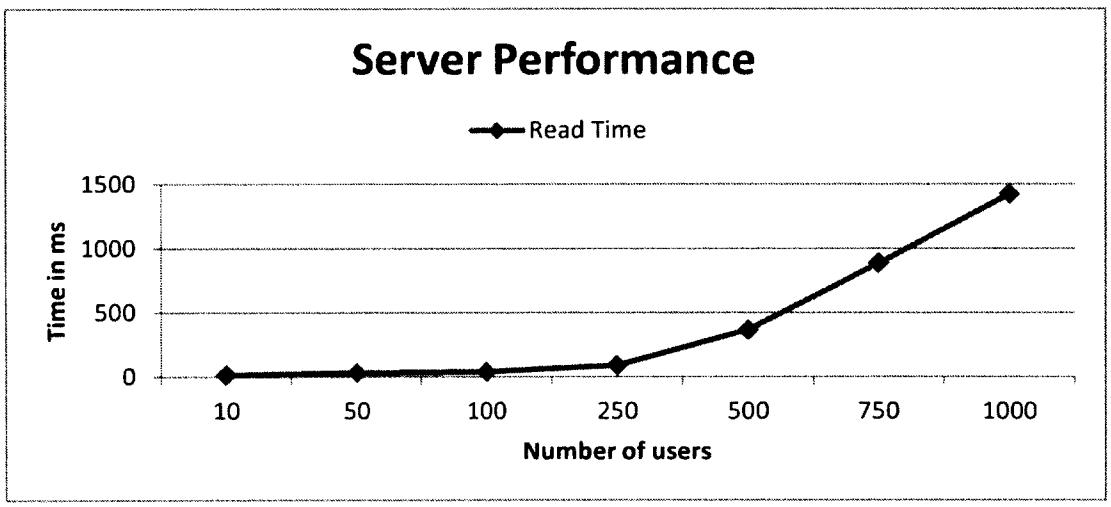


Figure 4.11 SDSA Web Server Performance Test Results – Read Data

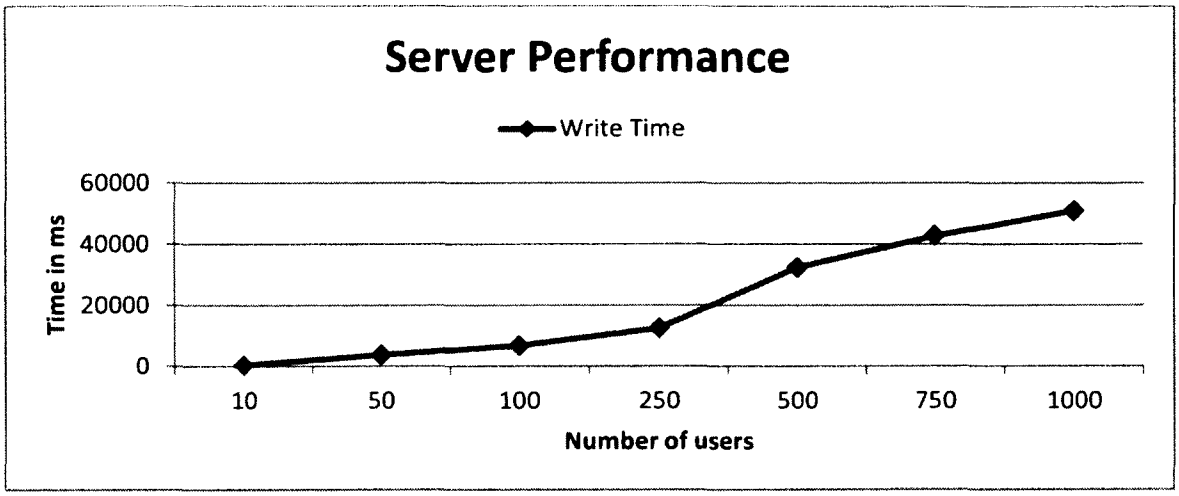


Figure 4.12 SDSA Web Server Performance Test Results – Write Data

We found that the read time remains unaffected when 100 users request data access concurrently. The message read time increases by 55.2% when number of users grows to 250. The time continuously increases as the number of users grow beyond 500. Even with 1000 users concurrently requesting read data access, the time taken to read a message is just 1.42 seconds. We observe that similar to read performance, the write time performance degrades linearly with growing number of users. Although, the performance degradation with write data operation is much higher than the read operation, due to the nature of write operations and limitations of underlying hardware of the server. Since we are using only single core CPU, the growing number of concurrent write operations, that involves intensive disk I/O and SQL data-integrity checks, increases the CPU load up to 100%. This causes significant increase in the average time taken to perform write data operation. However, even under high load our server can handle over few thousand messages in an hour. The solution can easily be scaled up by using more powerful server machine, or hosting the Web API on more than one servers and load balancing the requests. Table 4.5 shows the test results. We assume criterion 13 to be fulfilled.

No. of users	10	50	100	250	500	750	1000
Read Time [ms]	14.87	32.34	39.8	88.9	367	885	1424
Write Time [ms]	384.5	3762.64	6789	12480.37	32382.41	42769.69	50923.14

Table 4.5 SDSA Web Server Performance Test Results

4.6 Summary of Validation against Criteria

Criterion	Description	Fulfilled
1	Easy to use solution	Yes
2	Platform independent solution	Yes
3	Using Standard cryptographic algorithms	Yes
4	Secure generation of public-private keys	Yes
5	Secure storage of private keys on server and the device	Yes
6	Data recovery in case of device theft	Yes
7	Securing data from employees at telecommunication provider	Yes
8	Securing data while it is at the server	Yes
9	Securing data while it is being viewed on the mobile device	Yes
10	Low data read/write time	Yes
11	Low on-device resource consumption	Yes
12	Support for Wi-Fi and 3G	Yes
13	Scalability of the solution	Yes

Table 4.6 Summary of Validation against Criteria

Chapter 5

Conclusion and Future Work

Using mobile devices for healthcare is an emerging field that can significantly improve the way in which EMRs are currently stored and accessed. It increases the availability of patients' medical data especially during field visits and POCT scenarios, which assists healthcare providers in making better decisions regarding patients' health, therefore enhancing the quality of patient-care. However, there are number of challenges in implementing mobile data sharing solutions before adoption by healthcare systems (see chapter 1).

Mobile data sharing solutions proposed in various research papers, as discussed in chapter 2, fail to fulfill security requirements, and do not use mandatory standards for exchanging medical data. Most solutions assume encrypted data on the mobile device is protected. However, encryption just provides data-confidentiality, and does not provide data privacy. Another critical problem with these solutions is the way in which cryptography keys are managed. The keys are either stored on the mobile device or an additional hardware such as another mobile device, portable flash memory, and a key fob, is used to store the keys. Storing keys on the device introduces the risk of keys being stolen. Using additional hardware to provide keys can be a costly solution, especially when deployed for

organizations with large number of users. It also involves additional responsibility of securing the hardware from theft or loss.

Our proposed model has identified various vulnerable locations in mobile data-communication infrastructure that could possibly cause security breaches and compromise sensitive data. We have designed a low cost secure data sharing architecture called SDSA that addresses the aforementioned challenges. SDSA provides end-to-end data security during transmission, on the device, and the server. It consolidates data into a central repository located at server side behind a secure firewall, and uses standard security mechanisms such as authentication and encryption to provide data confidentiality and privacy. SDSA only allows registered devices to be used for accessing data. It performs two-step authentication that verifies the user and the device identity. By using simple username-password based authentication, SDSA allows integration with user management applications such as Active Directory. This allows organizations to categorize users in different groups and effectively control data access permissions for individual groups. SDSA divides the device's API key (used for device authentication) into two parts. The mobile device stores only one of the two parts, and therefore the API key remains secure in case of loss or theft.

Use of open standards such as XML, HTTPS, and SOAP ensure interoperability between different components, and allow integration with EMR applications. Various usability and technical criteria were designed to evaluate the security and feasibility of this work. A prototype application customized for the POCT scenario was built, and demonstrated that our solution meet all the requirements. Our solution outperforms data sharing solution proposed in [20] in terms of required computation resources and time taken

to read/write data. In addition to general data sharing, our solution has the ability to connect with EMR applications and retrieve patients' lab results collected from various facilities.

5.1 Limitation(s) of Proposed Work

1. SDSA allows users to securely access data by establishing a session with the remote server. Data access in connected mode enables organizations to keep complete control of the data usage and ensure that it is being accessed by authorized users. However, this makes data access limited to network availability, and data cannot be accessed in disconnected mode. In order to mitigate the downtime, SDSA supports both Wi-Fi and 3G network connectivity, which increases network availability.
2. HL7 specifications require all the mandatory data fields to be filled before data can be exchanged. This introduces an overhead in terms of message size. With our current implementation, the minimum message size would always be 4KB. Since we have implemented only few HL7 data fields, our prototype currently limits the usability to currently available fields. This limitation can be easily eliminated by including more number of data fields, as needed.

5.2 Future Work

A desirable feature in any data sharing solution is the ability to operate in disconnected mode. The increase in availability of Wi-Fi and 3G technologies have made it possible to get Internet connectivity almost everywhere, especially in developed countries. However, during field visits, healthcare providers may need to access data in rural areas with no network connectivity. A possible solution to this could be allowing user to temporarily download part of the data, while the network connectivity is available, and access it in

disconnected mode later. However, it would be impossible for organizations to control user authentication and track data access in disconnected mode. We would further research and discover possibilities of an effective offline authentication mechanism that would ensure data privacy.

We would like to extend our prototype to support connectivity with portable medical devices such as glucometer, and blood pressure monitors. This would benefit healthcare providers by enabling them to use mobile devices for conducting tests in real time, and directly storing the results as EMRs. Integration with medical devices involves compatibility and security issues. Few medical devices are currently available that support mobile connectivity. These devices use proprietary software applications and do not allow integration with other data sharing solutions.

In addition to the iOS client application, we have deployed our data sharing solution on an Android based Samsung Galaxy Tablet 10.1, by using Adobe's Flex framework [72]. Flex allows building mobile applications for iOS, Android, and Blackberry platform, by using single source code. We would conduct performance testing on Flex mobile applications, and compare it with native client applications. We would also like to deploy our prototype in an actual hospital/clinic, and further understand specific requirements to enhance the usability of our solution.

Bibliography

[1] Charles Newark-French, "Flurry: Time Spent On Mobile Apps Has Surpassed Web Browsing," Flurry, Inc., 20 June 2011. [Online]. Available: http://techcrunch.com/2011/06/20/flurry-time-spent-on-mobile-apps-has-surpassed-web-browsing/. [Accessed 27 June 2011].

[2] Tomi Ahonen, "Analysis: Record 80 Million Smartphones sold in 3Q 2010," 22 November 2010. [Online]. Available: http://www.brightsideofnews.com/news/2010/11/22/analysis-record-80-million-smartphones-sold-in-3q-2010.aspx. [Accessed 27 June 2011].

[3] "How smartphones and tablets are taking over," Gartner, 16 April 2011. [Online]. Available: http://www.techradar.com/news/mobile-computing/how-smartphones-and-tablets-are-taking-over-942724. [Accessed 27 June 2011].

[4] Salesforce Inc, [Online]. Available: http://www.salesforce.com/. [Accessed 27 June 2011].

[5] Square Inc, [Online]. Available: https://squareup.com/. [Accessed 27 June 2011].

[6] "Smartphones, Tablets Seen Boosting Mobile Health," 29 July 2010. [Online]. Available: http://www.pcworld.com/businesscenter/article/202224/smartphones_tablets_seen_boosting_mobile_health.html. [Accessed 29 June 2011].

[7] P. Tarasewich, R. C. Nickerson and M. Warkentin, "Issues in Mobile E-Commerce," Communications of the Association for Information Systems, 2002.

[8] B. Arunachalan and J. Light, "Middleware Architecture for Patient Care Data Transmission using Wireless Networks," Honolulu, Hawaii, USA, 2007.

[9] "Norton Seems to Make Your PC Slow," [Online]. Available: http://www.computergeeksonline.net/spyware/norton-your-pc-Running%20Slow.asp. [Accessed 30 October 2010].

[10] J. Oberheide, K. Veeraraghavan, E. Cooke, J. Flinn and F. Jahanian, "Virtualized In-Cloud Security Services for Mobile Devices," in *The First Workshop on Virtualization in*

Mobile Computing, Breckenridge, CO, USA, 2008.

[11] G. PortoKalidis, P. Homburg, N. F. Dale, K. Anagnostakis and H. Bos, "Protecting Smart Phones by Means of Execution Replication," Amsterdam, 2009.

[12] W. Yu, "The Network Security Issue of 3G Mobile Communication System Research," pp. 373-376, 2010.

[13] G. Singaraju and B. Hoon Kang, "Concord: A Secure Mobile Data Authorization Framework for Regulatory Compliance," in *Large Installation System Administration (LISA)*, 2008.

[14] J. Claeassens, B. Preneel and J. Vandewalle, "How Can Mobile Agents Do Secure Electronic Transactions on Untrusted Hosts?," in *ACM Transactions on Internet Technology (TOIT)*, 2003.

[15] B. A. Forouzan, Cryptography and Network Security, New York: McGraw Hill, 2008.

[16] N. Leavitt, "Mobile Phones: The Next Frontier for Hackers," *Technology News,* pp. 20-23, April 2005.

[17] A. Boukerche and Y. Ren, "A Security Management Scheme Using a Novel Computational Reputation Model for Wireless and Mobile Ad hoc Networks," in *Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*, Vancouver, BC, Canada, 2008.

[18] B. Halpert, "Mobie Device Security," in *1st annual conference on Information security curriculum development*, Kennesaw, GA, USA, 2004.

[19] "Health Level Seven (HL7)," [Online]. Available: www.hl7.org. [Accessed 18 May 2011].

[20] T. Matsunaka, T. Warabino and Y. Kishi, "Secure Data Sharing in Mobile Environments," Fuijimino, Saitama, Japan, 2008.

[21] C. Lassenius and T. Soininen, "Constructive Research," [Online]. Available: www.cs.uta.fi/~TKOPS407/sd-seminar-19-9-2007.pdf. [Accessed 12 October 2011].

[22] Apple, "iOS," Apple Inc., [Online]. Available: http://www.apple.com/ios/. [Accessed 15 May 2011].

[23] T. Lewis, "ECG Notes is an impressive ECG reference guide, but not designed for beginners," 19 July 2011. [Online]. Available: http://www.imedicalapps.com/2011/07/ecg-notes-highly-detailed-ecg-reference-guide/. [Accessed 20 August 2011].

[24] "Stay connected to your Health records on the GO.," MobileReflex, [Online]. Available: http://www.mobilereflex.com/services/hreflex.html. [Accessed 15 October 2011].

[25] B. Edwards, "BodyMedia is using wearable sensors to fight the obesity epidemic one step at a time," 25 July 2011. [Online]. Available: http://www.imedicalapps.com/2011/07/bodymedia-wearable-sensors-fight-obesity-epidemic-step-time/. [Accessed 20 August 2011].

[26] "electronic Point Of-Care," [Online]. Available: http://www.itacs.uow.edu.au/cear/ehealth/ePOC/. [Accessed 07 June 2011].

[27] P. Eklund and J. Sargent, "ePOC: Mobile Clinical Information Access and Diffusion in Ambulatory," in *10th Australian Document Computing Symposium*, Sydney, 2005.

[28] R. Poropatich, H. H. Pavliscsak, J. Rasche, C. Barrigan, R. A. Vigersky and S. J. Fonda, "Mobile Healthcare in the US Army," in *Wireless Health*, San Diego, USA, 2010.

[29] S. A. Becker, R. Sugumaran and K. Pannu, "The Use of Mobile Technology for Proactive Healthcare in Tribal Communities," in *Annual National Conference on Digital Government Research*, 2004.

[30] "ARTEMIS," 01 January 2004. [Online]. Available: http://www.srdc.metu.edu.tr/webpage/projects/artemis/. [Accessed 06 June 2011].

[31] "Triage," [Online]. Available: http://en.wikipedia.org/wiki/Triage. [Accessed 29 July 2011].

[32] S. P. McGrath, E. Grigg, S. Wendelken, G. Blike, M. De Rosa, A. Fiske and R. Gray, "ARTEMIS: A Vision for Remote Triage and Emergency Management Information Integration," [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.128.4959. [Accessed 21 June 2011].

[33] R. G. Jahns, "Will smartphone apps become the killer application of the mHealth market?," October 2010. [Online]. Available: https://www.research2guidance.com/will-smartphone-apps-become-the-killer-application-of-the-mhealth-market. [Accessed 18 June 2011].

[34] "How does Anti-Virus Softwares Work?," [Online]. Available: http://www.antivirusworld.com/articles/antivirus.php. [Accessed 15 November 2010].

[35] "Disk Encryption Hardware," [Online]. Available: http://en.wikipedia.org/wiki/Disk_encryption_hardware. [Accessed 15 August 2011].

[36] Hagai Bar-El, "Security Implications of Hardware vs. Software," [Online]. Available: http://hbarel.com/publications.htm. [Accessed 21 September 2011].

[37] "Vendor Lock In," [Online]. Available: http://en.wikipedia.org/wiki/Vendor_lock-in. [Accessed 15 August 2011].

[38] M. D. Corner and B. D. Noble, "Protecting Applications with Transient Authentication," in *Mobile systems, applications and services*, Ann Arbor, 2003.

[39] M. D. Corner and B. D. Noble, "Zero Interaction Authentication," in *The Annual International Conference on Mobile Computing and Networking*, Ann Arbor, 2002.

[40] R. L. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public- Key Cryptosystems," pp. 120-126, 1978.

[41] J. Light and D. David, "An Efficient Security Algorithm in Mobile Computing for Resource Constrained Mobile Devices," in *4th ACM symposium on QoS and security for wireless and mobile networks*, Vancouver, 2008.

[42] J. Heitala, "Hardware versus Software," SANS Whitepaper, 2007.

[43] "What Security Professionals Think about Encryption," 24 February 2006. [Online]. Available: http://www.csoonline.com/article/214971/what-security-professionals-think-about-encryption. [Accessed 01 July 2011].

[44] B. A. Forouzan, Cryptography and Network Security, McGraw Hill.

[45] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," [Online]. Available: www.cryptosoft.de/docs/Rijndael.pdf. [Accessed 21 October 2011].

[46] W. Stallings, Cryptography and Network Security Principles and Practices, Upper Saddle River: Prentice Hall, 2005.

[47] "TRANSMISSION CONTROL PROTOCOL," 1981. [Online]. Available: http://trac.tools.ietf.org/html/rfc793. [Accessed 11 October 2011].

[48] R. Rivest, "The MD5 Message-Digest Algorithm," April 1992. [Online]. Available: https://tools.ietf.org/html/rfc1321. [Accessed 21 October 2011].

[49] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," September 2011. [Online]. Available: http://tools.ietf.org/html/rfc3174. [Accessed 20 October 2011].

[50] D. L. Evans, P. J. Bond and A. L. Bement, "The Keyed-Hash Message Authentication Code," Federal Information Processing Standards Publication, 2002.

[51] S. Kelly and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec," 2007. [Online]. Available: http://www.ietf.org/mail-archive/web/ietf-announce/current/msg03683.html. [Accessed 02 November 2010].

[52] J. Quinn, "Health Level Standard Specifications," Health Level Seven, 1999.

[53] "Active Directory," [Online]. Available: http://en.wikipedia.org/wiki/Active_Directory. [Accessed 20 March 2011].

[54] Microsoft Corporation, "Active Directory Benefits for Smalled Enterprises," 2004. [Online]. Available: http://www.techrepublic.com/whitepapers/active-directory-benefits-for-smaller-enterprises/177983. [Accessed 02 November 2011].

[55] "Secure Random Numbers," Windows Developer Magazine, [Online]. Available: http://erngui.com/articles/rng/index.html. [Accessed 21 March 2011].

[56] *Advanced Encryption Standard (AES),* National Institutue of Standards and Technology, 2001.

[57] "BlackBerry Enterprise Solution: Security Technical Overview," Blackberry, [Online]. Available: http://docs.blackberry.com/en/admin/deliverables/16648/index.jsp?name=Security+Technical+Overview+-+BlackBerry+Enterprise+Server5.0.2&language=English&userType=2&category=BlackBerry+Enterprise+Solution+Security&subCategory=. [Accessed 12 October 2011].

[58] "iCloud," Apple Inc., [Online]. Available: http://www.apple.com/icloud/. [Accessed 15 September 2011].

[59] "Point-of-care testing," [Online]. Available: http://en.wikipedia.org/wiki/Point-of-care_testing. [Accessed 5 October 2011].

[60] "About Us," Northern Health, [Online]. Available: http://www.northernhealth.ca/AboutUs.aspx. [Accessed 06 October 2011].

[61] "What is service-oriented architecture?," [Online]. Available: http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html. [Accessed 05 October 2011].

[62] "SOAP Introduction," [Online]. Available: http://www.w3schools.com/soap/soap_intro.asp. [Accessed 05 October 2011].

[63] "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," [Online]. Available: http://www.w3.org/TR/soap12-part1/#intro. [Accessed 05 October 2011].

[64] "Mirth Connect," [Online]. Available: http://www.mirthcorp.com/products/mirth-connect. [Accessed 09 October 2011].

[65] "BizTalk Server," Microsoft, [Online]. Available: http://www.microsoft.com/biztalk/en/us/accelerator-hl7.aspx. [Accessed 13 October 2011].

[66] "Encrypting Configuration Information Using Protected Configuration," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/53tyfkaw.aspx. [Accessed 13 October 2011].

[67] "SecureString Class," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/system.security.securestring.aspx. [Accessed 10 October 2011].

[68] "SudzC," [Online]. Available: http://sudzc.com/. [Accessed 05 October 2011].

[69] "Medical Office Information System (MOIS)," AIHS, [Online]. Available: http://www.aihs.ca/products/. [Accessed 5 October 2011].

[70] "RNGCryptoServiceProvider Class," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/system.security.cryptography.rngcryptoserviceprovider.aspx. [Accessed 05 October 2011].

[71] "soapUI," SmartBear Software, [Online]. Available: http://www.soapui.org/. [Accessed 02 November 2011].

[72] "Flex," Adobe, [Online]. Available: http://www.adobe.com/products/flex.html. [Accessed 02 November 2011].

[73] "SQL Injection," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/ms161953.aspx. [Accessed 01 October 2011].

[74] "Keychain Services Programming Guide," Apple Inc. , [Online]. Available: http://developer.apple.com/library/mac/#documentation/Security/Conceptual/keychainServConcepts/01introduction/introduction.html. [Accessed 11 October 2011].

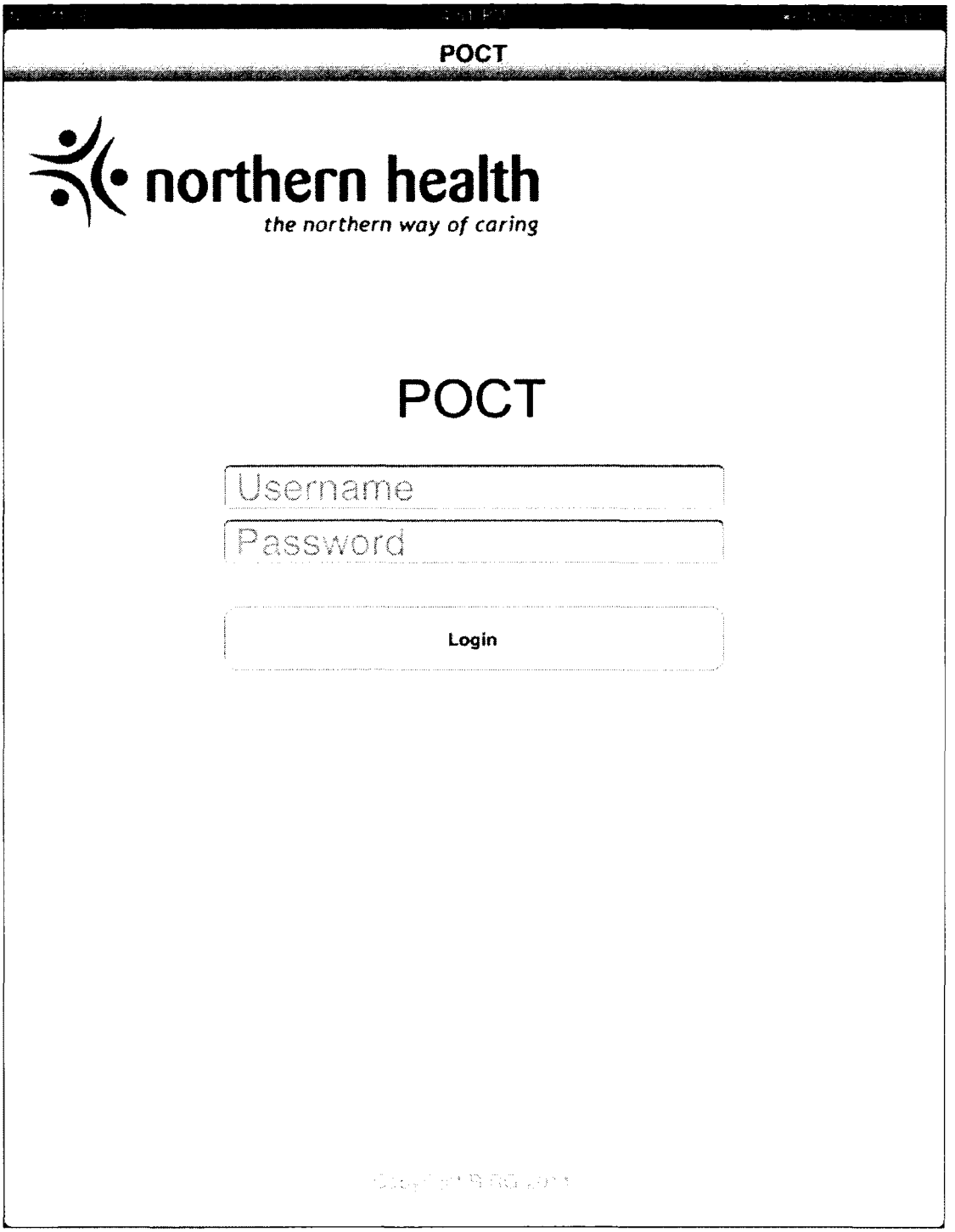
Appendix 1

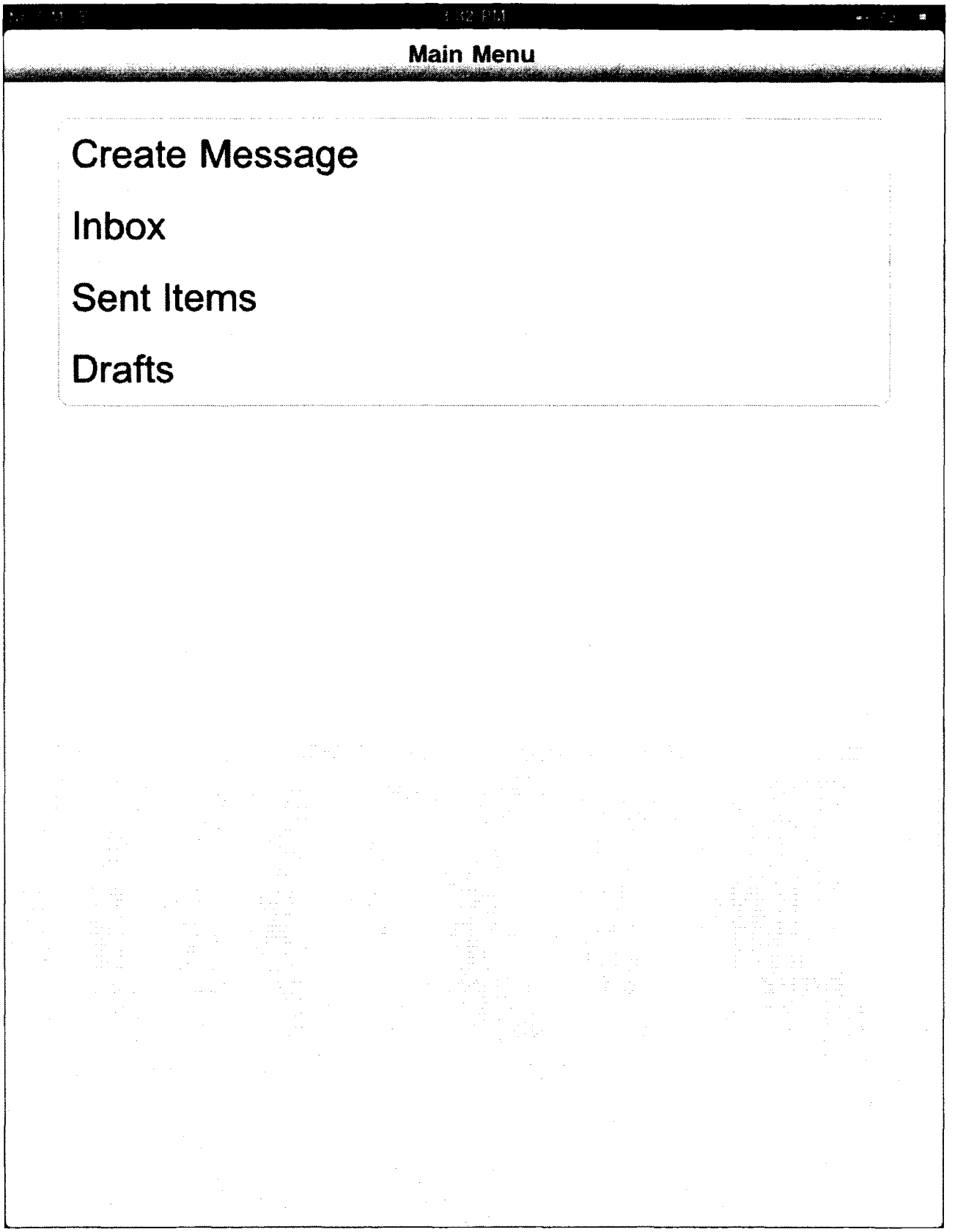
XML Message Format using HL7 version 2.3.1

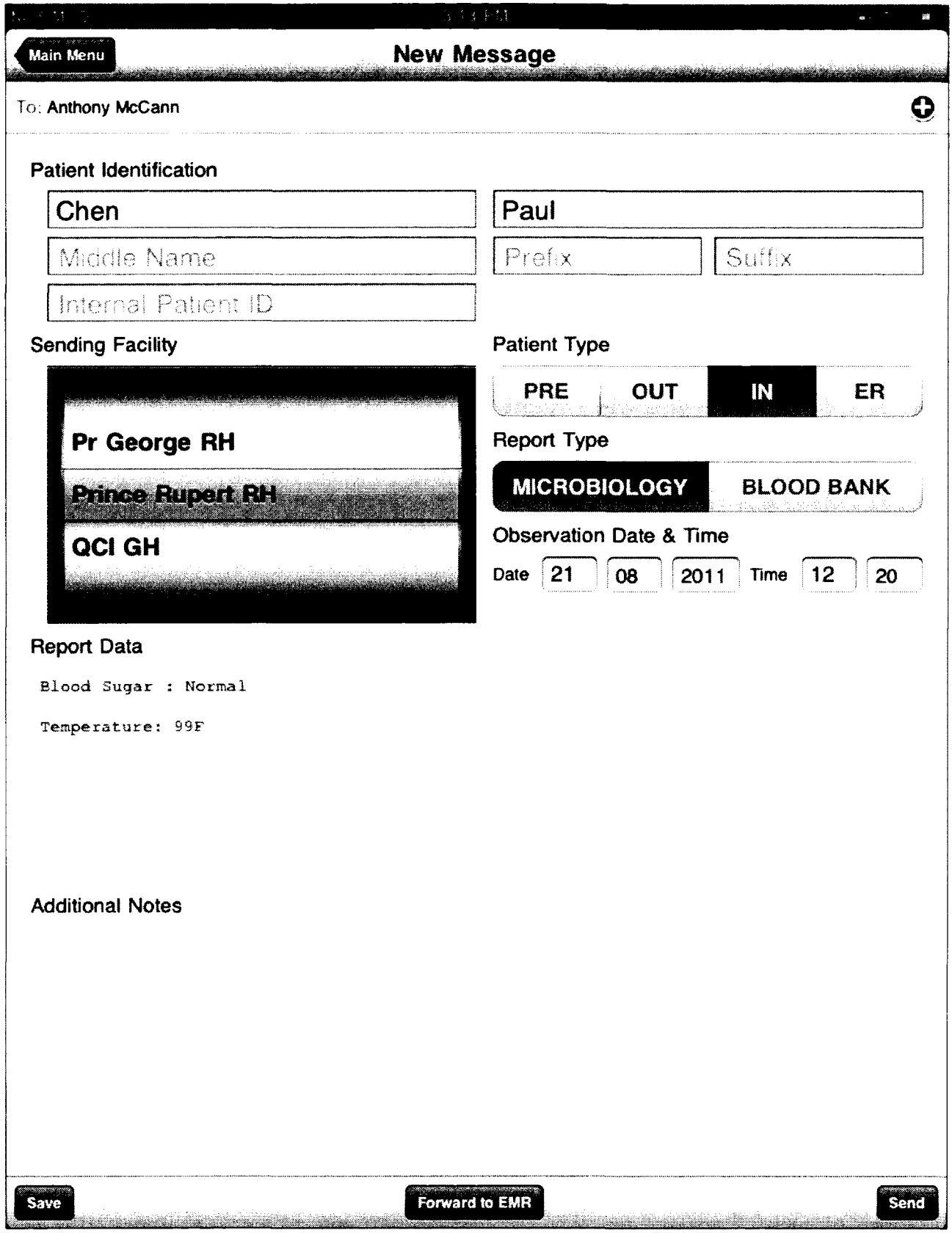
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<message>
        <MSH>
                <sendingApplication></sendingApplication>
                <sendingFacility></sendingFacility>
                <messageDateTime></messageDateTime>
                <messageType></messageType>
                <messageControlID></messageControlID>
                <processingID></processingID>
                <versionID></versionID>
        </MSH>
        <PID>
                <internalPatientID></internalPatientID>
                <patientName></patientName>
        </PID>
        <PV1>
                <patientClass></patientClass>
        </PV1>
        <NTE>
                <comments></comments>
        </NTE>
        <ORC>
                <orderControl></orderControl>
        </ORC>
        <OBR>
                <setID></setID>
                <placerOrderNumber></placerOrderNumber>
                <fillerOrderNumber></fillerOrderNumber>
                <universalServiceID></universalServiceID>
                <observationDate></observationDate>
        </OBR>
        <!-- multiple OBX segments are permitted. We only support the 'TX' type -->
        <OBX>
                <setId></setId>
                <Type></Type>
                <observation></observation>
        </OBX>
```

```
        <ZDR>
                <provider>
                        <providerMnemonic></providerMnemonic>
                        <lastName></lastName>
                        <firstName></firstName>
                        <middleName></middleName>
                        <suffix></suffix>
                        <title></title>
                        <degree></degree>
                        <providerCategoryCode></providerCategoryCode>
                </provider>
        </ZDR>
</message>
```

iOS Client Application Screenshots




Main Menu
Create Message
Inbox
Sent Items
Drafts


Main Menu
New Message
To: Anthony McCann
Patient Identification
Chen
Paul
Middle Name
Prefix
Suffix
Internal Patient ID
Sending Facility
Pr George RH
Prince Rupert RH
QCI GH
Patient Type
PRE
OUT
IN
ER
Report Type
MICROBIOLOGY
BLOOD BANK
Observation Date & Time
Date 21 08 2011 Time 12 20
Report Data
Blood Sugar : Normal
Temperature: 99F
Additional Notes
Save
Forward to EMR
Send

Inbox

TEST, TROY - MB - 7/25/2011 11:46:51 AM

	TEST, TROY		**NH#:**
	Sex:	**Age:**	**DOB:**
Mills MH	**Admit Date:**		
	Healthcare#:		
	Ordering:		
	Copies To:		

TEST, TROY 11-0785-44

ROUTINE CULTURES

PROCEDURE: Wound Culture
SOURCE: Wound COLLECTED: 09/16/2010 09:44 PDT
BODY SITE: Arm STARTED: 09/16/2010 14:57 PDT
ACCESSION: 24-10-259-8002

STAINS / PREPARATIONS

Gram Stain Report
Verified:09/16/2010 14:57 PDT
No organisms seen.

FINAL REPORT

Final Report
verified:09/16/2010 14:59 PDT
No growth at 24 hours.
No growth at 48 hours.