Implementation of a Secured Data Communication between Heterogeneous Systems using Web Services Case Study: Kumasi Polytechnic

by

EVANS KOTEI

A Thesis submitted to Department of Computer Science

Kwame Nkrumah University of Science and

Technology

in partial fulfilment of the requirements for the degree

of

MASTER OF PHILOSOPHY: INFORMATION TECHNOLOGY

Institute of Distance Learning (IDL)

JUNE 2016

Declaration

I, Evans Nana Agyei Kotei, declare that this submission is my own work towards the MPhil and that to the best of my knowledge, it contains no material previously published by another person nor material which has been accepted for the award of any other degree of the University, except where due acknowledgement has been made in the text.

Evans Nana Agyei Kotei	.....	.....
PG8310612	Signature	Date

Certified by:

Dr. Michael Asante	.....	.....
Supervisor	Signature	Date

Certified by:

Dr. Hayfron Acquah		
	.....	.....
Head of Computer Science	Signature	Date

Abstract

Web services are believed to be the future of distributed applications since they access a little resource from the host machine to operate. Though many developers have bought into this idea, the development of web services still needs a wide implementation and deployment (Du , 2004). Organizations are still finding it difficult to grasp its make up and deployment. Many organizations, still depend on partners infrastructure for data processing and its transfer. For example with respect to our case study (Kumasi Polytechnic), school fees data of students takes a long time before the institution acknowledges receipt. This is due to the fact that the banking institution involved has to do some internal reconciliations. In actuality this reconciliations do not benefit the institution but they bare its consequence in the area of student registrations.

In transferring or sending real time data, web service developers claim web services are the smartest way (Du , 2004), but customers would want to know whether it is really what it is meant to be. Its performance coupled with high security is very key to the customers who would want to implement such services. In this case web service providers need a well tested framework before the actual commercialization of the web services. For the past years, the Kumasi Polytechnic Institute had had many challenges with the the real time access to school fees data for other

process. Transactions sent over raw the hyper text transfer protocol(http) are susceptible to common attacks such as the man-in-the-middle attack. When this attack occurs, the attacker will be able to retrieve important messages from the http request and later use it against the real person the message was meant for. This is a main problem of raw http.

In this thesis, we showcase a secured way of transferring highly sensitive data through Hyper Text Transfer Protocol (HTTP) by implementing two kinds of security levels i.e. the Secure Socket Layer and the a single key data encryption algorithm. The results proved that encrypting data over the internet and also encrypting the transfer protocol has no significant effect of the data that is transfered.

Contents

Declaration ii

Abstract iii

List of Tables ix

List of Figures ix

Acknowledgements xii

Dedication xiii

1 Introduction 1

1.1 Background of Study 1

1.1.1 Extensible Markup Language (XML) 3

1.1.2 Web Service Definition 4

1.1.3 Service Oriented Architecture (SOA) 5

1.1.4 Universal Description Discovery ana Integration (UDDI) 6

1.1.5 Web Service Description Language (WSDL) 6

1.1.6 Simple Object Access Protocol (SOAP) 7

1.2 Problem Statement 7

1.3 Motivation 8

1.4 Objective 8

1.5 Research Questions 9

1.6 Significance of study 9

1.7 Methodology 9

1.8 Scope 10

1.9 Organization of Thesis 10

2 Literature Review 11

2.1 Introduction 11

2.1.1 Background 11

2.1.2 Distributed Computing 12

2.2 Service-Oriented Architectures 16
2.2.1 Definition and Concepts 18
2.3 Web Service 20
2.3.1 Web Service Technology 21
2.3.2 W3C Web Service Architecture 21
2.3.3 Web Service Description Language (WSDL) 23
2.3.4 Structure of WSDL 23
2.3.5 Work-flow of Web Services 26
2.3.6 Web Services Limitations 27
2.3.7 Extensible Markup Language (XML) 27
2.3.8 XML and Documentation 28
2.3.9 Universal Description, Discovery and Integration, (UDDI) 28
2.3.10 Simple Object Access Protocol, SOAP 29
2.3.11 Representational State Transfer (REST) 31
2.3.12 SOAP vs. REST services 33
2.4 Web Service Security 34
2.5 Development Tools and Technologies 34
2.5.1 PostgreSql Database 35
2.6 Service Developmental Strategies 36
2.6.1 Code First Approach 37
2.6.2 Contract First Approach: WSDL 37
2.7 Summary 39
3 Methodology 41
3.1 Introduction 41
3.2 Test Case Development 41
3.2.1 Deploying over Java Client 45
3.2.2 Deploying over PHP Client 46
3.3 Existing System 46
3.4 Proposed System 47

3.4.1 Message Ciphering 48
3.4.1.1 Ciphering Algorithm 50
3.4.2 Message Deciphering 50
3.4.2.1 Deciphering Algorithm 51
3.5 Web Service Performance Test 52
3.5.1 Test Suite:soapUI 52
3.5.2 Functional Testing 52
3.5.3 Performance Testing 53
3.5.3.1 Number of Transactions 53
3.5.3.2 Transaction Size 56
3.5.3.3 Load Test 60
3.5.4 Testing Over a Network (LAN) 62
3.6 Security Implementation 63
3.6.1 Secure Socket Layer (SSL) Implementation 63
3.6.2 RSA algorithm for the Web Service Certificate Generation 64
4 Results and Discussion 68
4.1 Design, Results, Analysis and Discussion 68
4.2 Functional Test 68
4.3 Non Functional Test 70
4.3.1 Load Test 70
4.3.2 Transaction Size 71
4.3.3 Testing Over a Network (LAN) 74
4.3.4 Security Test 75
4.3.4.1 HTTP Encrypted Transaction Mode 75
4.3.4.2 HTTPS Encrypted Transaction Mode 77
4.3.4.3 HTTP Raw Transaction Mode(Existing System) 79
4.4 Summary 81
5 Conclusion 82
5.1 Conclusion 82

5.2 Summary of Findings....82
5.2.1 Support for Heavy Load....83
5.2.2 Response Time....83
5.2.3 Managing Internet Resource....83
5.2.4 Secured....84
5.3 Recommendation and Future Work....84
Appendix....88

List of Tables

2.1 Elements of Service Contract (w3schools , 2014) 38

3.1 HTTP with Raw Message 53

3.2 HTTP and HTTPS with Encrypted Message 54

3.3 Transactions per Minute for Same Character (HTTP) 55

3.4 Transactions per Minute for Mixed Characters (HTTP) 56

3.5 Transactions per Minute for Same Characters-HTTPS medium 57

3.6 Transactions per Minute for Mixed Characters (HTTPS) 58

3.7 Thread Load for HTTP transmitting Raw Message 59

3.8 Thread Load Results for Encrypted Transactions over HTTP and HTTPS 60

3.9 Number of Transactions per Minute on LAN 61

4.1 Mean Number of Transactions per Minute on LAN 74

List of Figures

2.1 A time line of distributed computing taken from (Leitner , 2007) 13

2.2 An example of client server application(Leitner , 2007) 14

2.3 CORBA architectural model 15

2.4 MOM architectural model 16

2.5 Service Oriented Architecture 17

2.6 Main principles of Service-Oriented Architecture and their relations (Felipe , 2010) . 19

2.7 The General Process of Engaging a Web Service (Felipe , 2010) 21

2.8 WSDL extensibility 23

2.9 Work-flow of Web Services 26

2.10 Types of Registries and their relationships (Organization for the Advancement of Structured Information Standards , 2014) 29

2.11 A SOAP Envelope 30

2.12 A typical SOAP Envelope 30

2.13 SOAP Architecture (Du , 2004) 31

2.14 32

2.15 Web service technologies comparison.(Felipe , 2010) 33

3.1 Service Interface 41

3.2 Service Implementation Class 42

3.3 Service Publisher Class 43

3.4 Auto Generated WSDL File 43

3.5 Java Client Code 44

3.6 Java Client Response 44

3.7 PHP Client Response 45

3.8 Single Key Encryption 48

3.9 Soap Fault Generated by soapUI (Error) 52

4.1a Validation Test a 68

4.1b Validation Test b 69

4.2 Thread Load 70

4.3 Transactions per Minute for Mixed Characters (HTTP) 71

4.4 Mixed Characters (HTTPS enabled service 72

4.5 Mean Number of Transactions per Minute 73

4.6 LAN Performance 74

4.7 Plain Logs showing Requested Files 75

4.8 Plain Logs of Request 76

4.9 Logs of Secured Service 77

4.10 Logs of Secured Service 77

4.11aClient Soap Request 79

4.11bResponse from HTTP 79

4.11cServer Soap Response 80

Acknowledgements

I am particularly grateful to the Almighty God for guiding me throughout this thesis. Many thanks to my Supervisor, Dr. Asante, for guiding and directing the success of this thesis. I am most grateful for his support during my stay at the Department of Computer Science. I feel honored to have collaborated with Mr. Allen Eben Tetteh of Department of Mathematics for multiple reasons.It was he who believed in me and supported me at difficult times. Thank you so much.

Dedication

I would like to dedicate this thesis to my beloved wife, Mrs. Kotei.

Chapter 1

Introduction

1.1 Background of Study

The bond between the Internet and the users have become great over the years since web pages have gone through a lot of changes from a time where the Internet mostly provided static pages to now where the internet is full of dynamic pages. The high increase of Internet users to day has compelled a lot of companies,businesses and organizations to move their services or products online.

In serving users well and also winning users loyalty on the Internet, Companies like Internet Service

Providers (ISPs) have created portals to integrate and classify their information services (Felipe , 2010) like news so that users could get access to any news around the world at a single place just to facilitate information retrieval. In the late 90s, saw the introduction of search engines that allows users to search for services and content from a variety of service providers that addressed their needs, thus reducing the influence and patronage of the portals. The internet has since received a tremendous growth in terms of technology and standards. These standards and technologies like XML,AJAX, web services have enabled companies to develop a wide range of media based or social components (e.g.: Facebook, YouTube, delicious.com etc). The way users and companies also interact with the Internet has changed over time because now even non-technical people can create content and share information among themselves and because of this, the Internet has become a

space where new services and content are continuously growing at a faster pace. Integrated services has also become common in the web community since several businesses and Government organizations have embraced the act of developing web services which are some times in the form of applications created on the fly out of programs and data that live on the Internet.

Since Internet came to existence, web-applications have played a pivotal role in the development of businesses and organizations by way of moving them from the traditional brick and mortar infrastructures to online infrastructure which are situated in different locations (Ramesh et al , 2003).

At the moment, software applications are previewed to content or data over the World Wide Web regardless of the programming languages they are written in. The Web Service technology insures a paradigm where two or several heterogeneous software applications share data among themselves. The data or information sharing is typically delivered through the Internet over the Hyper Text Transport Protocol (HTTP). By this the applications are sort of webified in order for the transfer to take place. Incorporating a web service into any software application enables the application to expose specific functionalities that are consumed by other software. Every web service that one develops must be reliable and its performance should be tried and tested to build the confidence of organizations and companies, that web services are reliable and that they can always rely on any published service that addresses their need rather than building a new system which will save them time and money.

Kumasi Polytechnic Institute has had many challenges with the sale of admission forms and subsequently resolving the payment of students tuition fees. Kumasi Polytechnic has several vendors that assist the institution in the sale of admission forms all over the country. These include

financial institutions and non-financial institutions like the post office and other governmental agencies.

This possess some sort of threat to Kumasi Polytechnic especially since the non monetary institutions sometimes fail to render proper accounts on the sale of admission forms. In addition to this students are made to queue every academic year for tuition fees receipt verification and clearance. The cashiers at the polytechnic go through this process in order to make sure that students pay their tuition fees before they are ”cleared” to register. This has been a menace to the entire student populace and the institution. The idea of web services could be channeled in a unified manner in building an integrated system that could facilitate the easy flow of some activities in the school.

1.1.1 Extensible Markup Language (XML)

This is a structured language that describes a set of regulations for presenting documents in a format that is readable to the user and the computer. Web service is the latex technology in distributed computing, based on XML standards and Internet protocols and also a powerful tool that facilitates communication and collaboration between business applications which were developed on different platforms and are also running on different resources to work as one. Extensible Markup Language, XML have been neglected by many developers in terms of its strength and capabilities. It is powerful tool such that its capabilities stem from documentations, development of databases, a medium of data or information exchange between heterogeneous systems etc.

1.1.2 Web Service Definition

This is a tool or technology that is used for data communication between applications through the use of Extensible Markup Language (XML) tags, JavaScript Object Notation (json) and network protocols like HTTP. These technologies come together to offer services in a more natural way where by there is a request of service and an offering of that service if that service is available. In actual sense web service(s) is/are method(s) or function(s) that is/are described by a WSDL and are made available or published via UDDI. Web services can be seen as the bench mark or the standard for integrating applications in order for them to communicate very easily based on its XML component. Web services unlike web pages do not have GUI connecting the sever and the client. They rather share the application logic, processes and data through the Internet or a network interface (Chandrasekar, 2003).

It is distributed system of loosely coupled applications whose backbone is the service oriented architecture (SOA) deployed over the HTTP. A typical example is Amazons Web Services (AWS). This infrastructural setup provides online services for other websites or client-side applications. The world wide web consortium (W3C) defines a web service as

"a software system designed to support interoperable software-to-software interaction over the Internet. It has an interface described in a machine-processable format (specifically Web Service Description Language (WSDL))",(Brown et al, 2004). This shows that once a web service is up and running, any other system or application can request for the services given the right access.

1.1.3 Service Oriented Architecture (SOA)

Web services operates on the Service-oriented Architectures (SOA) (Jones, 2205) which uses interoperability as its communication protocol and a broker-request architectures to facilitate exchanges of service. The Organization for the Advancement of Structured Information Standards (OASIS) (Leitner, 2007) (OASIS, 2006) defines SOA as paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.

SOA can also be defined as a form of technology architecture that adheres to the principles of service-orientation. Looking into Web service technology platform, SOA depicts the power to support and promote these principles of the entire business process and automation of an enterprise

(Leitner, 2007). SOA in detail has specific features which are listed below:

- loosely coupled - services are self-contained and self-managing. The number of necessary connections to systems outside of the service are minimal. Services have low representational, identity and communication protocol coupling (Papazoglou et al, 2006).
- defined by a service contract - services adhere to a communications and interface definition or to a service description,
- autonomous - services have the absolute control over the function that they realize,
- abstract - services hide all implementation details from the rest of the world,revealing only the service contract,
- reusable - services are intended for and promote reuse,
- simple services can be assembled and coordinated to build composite services (service composition) (Curbera et al,2003)(Michael et al , 2005)

- stateless - services do not have a state, and
- discoverable - services can be found and evaluated via external discovery or registry mechanisms.

A typical SOA architecture consist of three main actors. The Provider,the Broker and the Requester. In this scenario a service provider creates the services which is then made available to the service requester through the service broker(Simmonds , 2011). The service requester accesses the components of the service through the Universal Description, Discovery and Integration, UDDI which has all the information that the requetser needs.

1.1.4 Universal Description Discovery ana Integration (UDDI)

This contains all the needed information, parameter, and function about a published webservice to enable client invocation It enables service providers to showcase all their services in other for service requesters to find and consume those services. The UDDI has two main parts or attributes. Firstly, it has a registry of all the web service's meta data and secondly a set of Web S Description and the port type definitions for searching that registry (The Tutorials Point , 2014).

1.1.5 Web Service Description Language (WSDL)

This is the main language that the UDDI uses in its operations. It is commonly used in conjunction with XML data schema to serve a web service on the Internet. A service requester searching for a service to consume looks for the UDDI from the WSDL file for all the method that the service provider has served. The requester then uses a SOAP to connect to the specific function which it needs(**?**).

1.1.6 Simple Object Access Protocol (SOAP)

This is a protocol for exchanging messages written in XML. Its way of transferring data on a network is achieved in conjunction with the Hyper Text Transfer Protocol, HTTP(S). A extensive view on SOAP,UDDI,WSDL and SOA will be carried out in detail later in the next chapter.

1.2 Problem Statement

Web services are believed to be the future of web applications since they access a little resource from the host machine to operate. Though many developers have bought into this idea, the development of web services still needs a wide implementation and deployment (Du , 2004). Organizations are a still finding it difficult to grasp its make up and deployment. Many organizations, like the one cited in this thesis still depend on partners infrastructure for data processing. For example data on school fees delays a day or two before the finance office of the institution gets access to it. This is because their banking institution has to do some reconciliations which do not benefit the institution when it comes to student registration. This goes a long way to affect student registrations and some other pertinent activities in the school. In transferring or sending real time data, web service developers claim web services are the smartest way, but the requesters would want to know whether it is really what it is meant to be. Its performance is very key to the development of their businesses. In such a situation service providers need an above experimental proceedings before the actual commercialization is deployed. For the past years, the Kumasi Polytechnic Institute had had many challenges with the the real time access to school fees data for other process. If even they had one, their main concern is the level of security the system would be endowed with. Due to this, students are made to queue every academic year for receipt

verification. In other to solve some of this problems and unleash the capabilities and advantages of web services, this thesis was proposed.

1.3 Motivation

More than two decades ago saw the advent of Extensible Markup Language by a group of developers in association with the W3C (**?**). Since then few technologies have captured its capabilities for deploying systems that can be used in industries. More recently, many developers are beginning to unleash the potentials and power of the XML language in developing web service driven applications that could seamlessly connect with any other application (Singh, , 2004). Web services can be used to leverage different applications between the same or different companies instead of rebuilding them. This is able to remove all forms of platform or hardware issues since web services are platform independent. Web service is less expensive to implement since it takes less time to develop and also improving some of the service components which helps in saving time in terms of service adaption. The learning curve for developers can also be reduced as well when considering that it is not necessary to learn specific details behind the services. The risk involved in web services is mitigated since there are already tried and tested services available that can be reused. This reduce failures when developing new services. There is also easy adaptability since the configurations about integrated services can be changed easily. This allows easy and quick deployments.

1.4 Objective

This thesis addresses the following objectives:

- To use web service as a tool to send data between two heterogeneous applications
- To secure pertinent data (fees) before being transacted
- To measure the performance of the developed web services.
- This research is geared towards the development and deployment of a service which will seamlessly integrate the systems of Kumasi Polytechnic and its agents.

1.5 Research Questions

- How can the transportation medium of data be encrypted before sending data through it?
- How can fees data be secured before being sent over the internet?
- Would the performance and security of the new web services hinder the existing system?

1.6 Significance of study

This work addresses the performance issues associated with web services that organizations, companies and individuals are not aware of. The study has also proven that web services are very reliable, efficient and secured platform for two or more applications to communicate without any regards to the hardware or platform on which each was developed.

1.7 Methodology

The main web service developed in this thesis is Java based. In every system, performance measurement is very keen to ensure optimum usage of the system. In other to determine the performance of the web service created, this study took upon itself to measure some key properties

of the developed service. This was done to ensure the efficiency of the service. We demonstrated a simple web service with two clients notably Java and PHP clients as a test case before the main web service for sending school fees data was developed and deployed. Secured Socket Layer (SSL) was implemented to secure the web service. The tools and technologies used for this work are eclipse Integrated Development Environment,PostgreSql, Ubuntu 12.04 LTS. We proposed an experiment for testing the service functionally and non functionally with a simulator known as soapUI.. The service performance and its security was also tested.

1.8 Scope

This thesis is focused on web services and its implementation. The technology used is the Service Oriented Architecture (SOA). A SOAP based web service is developed for both the client and the server. In this thesis the RESTful architecture which is another SOA technology and similar to SOAP was not used for the service development but it was reviewed in the literature. Added to the above the thesis dwells the development of a secured communication system between Kumasi Polytechnic and all its financial agents.

1.9 Organization of Thesis

Chapter one in this thesis talks about the introduction and background of the study.Chapter two is the literature review. Literature in the area of this study are reviewed and summarized.The methodology employed in this study is presented in chapter three.The analysis and simulation results are presented in chapter four. Chapter five encompasses the conclusion and recommendations for further research.

Chapter 2

Literature Review

2.1 Introduction

There has been a considerable amount of previous academic research in this field. Web service is a self-contained, modular application built on a deployed network infrastructure including XML and HTTP. Its description uses pale standard for its description (WSDL), discovery (UDDI) and invocation (SOAP). Web services are widely known now and its base on its performance .

In this chapter is an introduction of Web services, and an overview tried and tested technologies in reliability and web service composition techniques (**?**). Then, a literature review of current reliable Web service systems and Web service composition is presented. Also a lot of accademic researchs, journals, videos, audio, books, electronic materials and articls relating to this research area have been reviewed.

2.1.1 Background

The Internet today is used for several purposes base on your needs and your request These web enabled applications are built using different software applications to generate HTML codes and their access is limited through web browsers or by using an application specific client (Ramesh et al , 2003). HTML and web server technologies are only for presentation and are not able to interract with othe applications. The introduction of web services has brought a change in the exchange of

information across the internet base on Internet standard and technologies. Web services are able to encapsulate applications and publish them as services in the form of XML. Web services has leverage the communication channel between programming languages there by ensuring interoperability of applications.

2.1.2 Distributed Computing

In recent years Main Frame computers and applications were considered to be the best solution for solving large scale data processing applications. Few years later saw the introduction of personal computers which could contain more applications as compared to the Main Frame era. Personal computers became more popular interms of cost and ownership and ease of application used. As the number of applications running on individual PCs grew up, communications between such application programs became more complex and it prvented application to application interaction. Soon after the introduction of personal computes, network computing which has gained much importance and enabling remote procedure calls (RPCs) over a network protocol called Control Protocol/Internet Protocol(TCP/IP) turned out to be a widely accepted way for application software communication. Network computing also faced a number of challenges since the software applications were running on different hardware platforms, operating systems, and different hardware networks needed to communicate with each other and share data. These challenges led to the introduction of Distributed Computing applications. **It can also be defined as an application that is made up of N(with N greater than 1) physically independent computers, but looks like one single coherent system to the user (Leitner , 2007) (?)**.

Since the introduction of Distributing Computing, many are the organizations that have benefited from it, interms of information technology. Newly developed and implemented systems have always

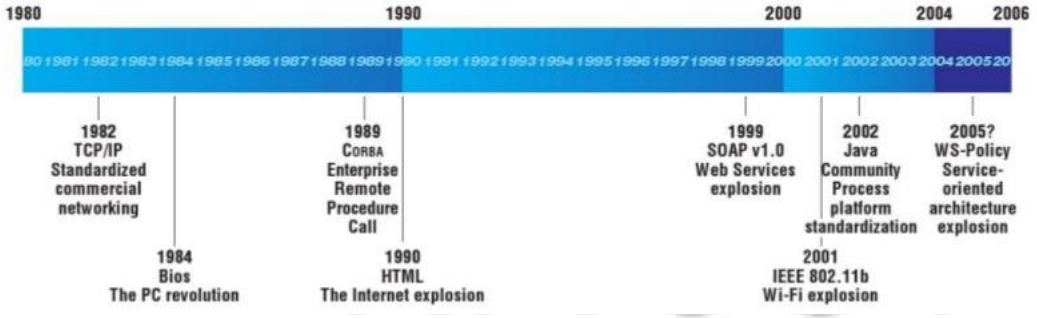

Figure 2.1: A time line of distributed computing taken from (Leitner , 2007)

been done with distributing computing in mind. Distributed computing existed for a while and Enterprise Application Integration (EAI) was born but it did not receive much petronage beacause it was hard to implement. The systems were developed on different applications and hardware platforms which were using various proprietory protocols and the number of applications that had to be integrated was growing day by day. The above difficulties gave birth to several distribution and integration technologies such as message-oriented middleware (Leitner , 2007) (Guruduth et al , 1999) (Gregor et al , 2003).

The Message Oriented Middleware rendered a smooth integration patterns such as Publish/Subscribe middleware systems for remote procedure calls (Leitner , 2007) (RPC) which massively simplified the task of writing distributed software systems. With the introduction of object oriented programming, the RPC middleware was expanded to distributd object middleware which had the tendancy to call objects on remote machines as if they were stand-in memory objects (**?**).

Distributing Computing comes in different technologies which includes Client/server application, CORBA, JavaRMI, MicrosoftCOM, DCOM and MOM. **Client Server Application** The client server architecture is of two parts. The first part is the upper tier which is made up of the presentation and business logics where as the other part (lower tier) is made up of the application and its

backend database. The server is a database that is mainly responsible for managing and retrieval of data. The client handles the business processing and provides the graphical user interface of the application Enterprise Resource Planning (ERP) is one application that is widely operated base on the client server architecture. The client application is installed on different multiple desktops and connected to central database system which is the server. Even though this architecture is widely used by organizations, it has some limitations which are listed below:

- To process complex transactions and business processes, a robust client system is needed.
- Security is always an issue since the client machines are always exposed to the outside world and that makes it vlnerable to hackers.
- Bigger bandwidth is needed to be able to process many calls to the server which can impose scalability restrictions.
- There is also difficulty in maintenance since each client machine would have to be dealt with individually.

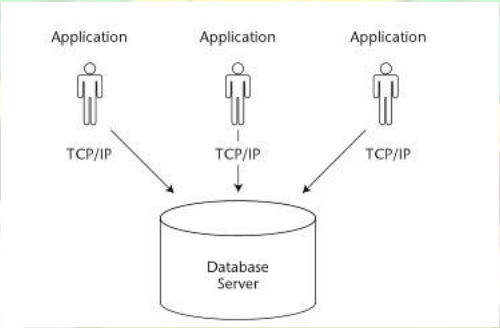

Figure 2.2: An example of client server application(Leitner , 2007)

Common Object Request Broker Application The common object request broker(CORBA), is an open standard developed by the Object Management Group (OMG) for enabling distributed

computing that supports a wide range of application environments. OMG ensures production and maintenance of framework specifications for distributd and interoperable object oriented systems.

CORBA is different from client server architecture because CORBA has certain features that the traditional client server architecture does not have.

CORBA is in two versions now, version 1.1 and 2.0. Version 1.1 focused on the creation of component level and portable object applications without interoperability whilst the version 2.0 ensures interoperability between distinct ORB vendors through the Internet Inter ORB Protocol(IIOP). This protocol actually explains the backgroud of the ORB. In CORBA's operations, the ORB is used as a communication bus that provides an avenue for tranporting requests and receiving acknowledgements irrespective of the location. ORB intercepts all requsts from the client and looks for the corresponding server that works on the request and lays its parameters,invoke its function and sends back the findings to the client. Another function of the ORB is to provide an interface for the CORBA services which facilitates the bulding of custom distributed application environment. The CORBA architecture consists of the following: **IDL**. The IDL spcifies the application boundaries and creates interfaces with the its clients. **ORB** It provides a communication bus that is used for sending and receiving request/response from the client or server It also provides interoperability in heterogeneous environment. Below is the CORBA architecture

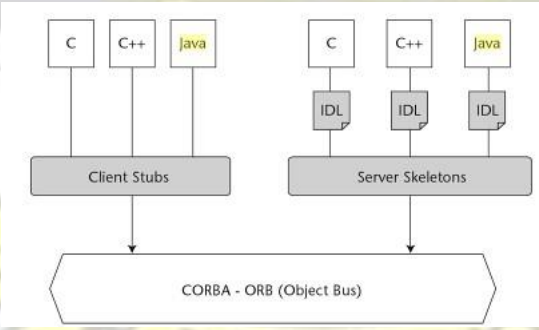

Figure 2.3: CORBA architectural model

Message Oriented Middleware This is a loosely coupled asynchronous communication model of which the client need not to be aware of its recepient or its functional parameters. Some of the widely known MOM -based technologies are SunONE Message Queue,IBMQseries,TIBCO, SonicMQ etc. This is a typical MOM architecture Despite all the advantages associated with the

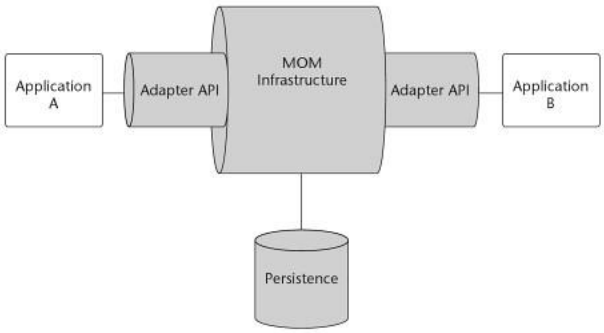

Figure 2.4: MOM architectural model

MOM based architecture, it has som challenges:

- The messaging format for application integration do not follow any standards but it is based on proprietary message format.
- Most MOM implementation has an API that talks directly to their core infrastuctures there by making portability with other applications very difficult.

2.2 Service-Oriented Architectures

This thesis report focuses on serviced-based applications but before anything else the background knowledge would be explored first. This will entail the familiarization with the web service architecture, service-oriented concepts and then construct a framework suitable for experiment. The services that make up these type of applications are not necessarily provided by the application

owner but can be provided by a third party. More recently there have been several advancement on how these type of applications are built. The most common framework is the Service- Oriented Architecture(The Open Group, , 2014). With SOA's, one is capable of developing systems written in different languages capable to connect and interract with on another. Every Service Oriented Architecture consist of three main actors namely the Provider,Broker and Requester. Here the service provider creates the services which is then made available to the service requester through the service broker(Simmonds , 2011). Figure 2.1 depict the architecture of an SOA.

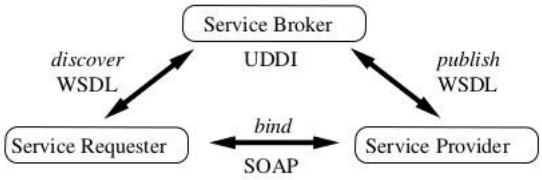

Figure 2.5: Service Oriented Architecture

In a web service, the Universal Disccription Discovery and Integration (UDDI) is used to acess the componets of the web service which is in the form of an XML. The UDDI gives description about the published web service and also helps in location it. It enables service providers to showcase all their services in other for service requesters to find and consume those services. The UDDI has two main parts or attributes. Firstly, it has a registry of all the web service's meta data and secondly a WSDL port type definitions for searching that registry (The Tutorials Point , 2014).

More on UDDI will be discussed later in this chapter.

The WSDL is an XML language used for data or information communications over a network. In this case it is the main language that the UDDI used in its operations. It is commonly used in conjunction with XML data schema to serve a web service on the Internet. A service requester

searching for a service to connect to reads or consume looks for the WSDL file which contains all the method that the service provider has served. The requester then uses a SOAP to connect to the specific function which it needs(**?**). An extensive discussion on WSDL will be carried out later in this chapter. The SOAP way of transferring data on a network is achieved in conjunction with the Hyper Text Transfer Protocol, HTTP(S). A broad view on SOAP will be mentioned later on in the chapter. Service-Oriented Architectures is one of the technologies that in computing now that is receiving a tremendous growth in the field of computer science or industry. It has also received a lot of attentionfrom researchers and practitioners. SOAs are considered as the next major step in distributed computing (Leitner , 2007) (**?**) by a big part of the research community today.

2.2.1 Definition and Concepts

The Organization for the Advancement of Structured Information Standards (OASIS) has recently published a reference model for Service-Oriented Architectures (Leitner , 2007) (**?**)which defines a SOA as follows: Service-Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains.citePhilip

SOA can also be defined as a form of technology architecture that adheres to the principles of service-orientation. Looking into Web service technology platform, SOA depicts the power to support and promote these principles of the entire business process and automation of an enterprise.(Leitner , 2007)

Looking at the two definitions above, the one from OASIS and Philip, they are not detailed and explanatory enough since SOA goes beyond the concept of a service built on top of an architecture. SOA in detail specific features which makes it a very powerful tool. These features are listed below:

- loosely coupled - services are self-contained and self-managing. The number of necessary connections to systems outside of the service are minimal. Services have low representational, identity and communication protocol coupling (**?**).
- defined by a service contract - services adhere to a communications and interface definition or to a service description,
- autonomous - services have the absolute control over the function that they realize,
- abstract - services hide all implementation details from the rest of the world,revealing only the service contract,
- reusable - services are intended for and promote reuse,
- composable - in order to promote reuse services are easily composable, i.e.
- simple services can be assembled and coordinated to build composite services (service composition) (**?**)(Michael et al , 2005)
- stateless - services do not have a state, and
- discoverable - services can be found and evaluated via external discovery or registry mechanisms.

The figure below depicts these properties, their relations and how each property strengthens and is strengthened by the others. All the elements, relations and constraints that makes up the SOA

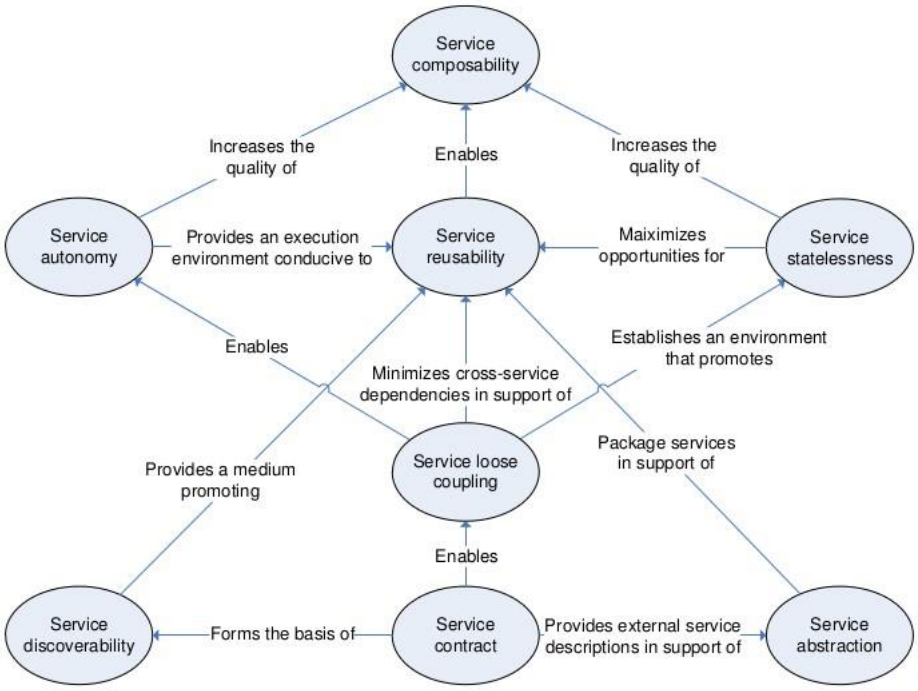

Figure 2.6: Main principles of Service-Oriented Architecture and their relations (Felipe , 2010)

architecture have been defined in the OASIS reference model.

2.3 Web Service

Web service uses XML tags and Java Script Object Oriented Notation(json) technologies and some network protocols for its operations. These technologies come together to offer services in a more natural way where by there is a request of service and an offering of that service if that service is available. In actual sense web service(s) is/are method(s) or function(s) that is/are described by a WSDL and are made available or published via UDDI. Web services can be seen as the bench mark or the standard for integrating applications in order for them to communicate very easily based on its XML component. Web services unlike web pages do not have gui connecting the sever and the client. They rather share the application logic, processes and data through the Internet or a network interface (Manoj et al , 2003). SOA sees a web service as an instance of the SOA and it can be

published, discovered and activated over a network using a SOAP message protocol framework in XML format (Du , 2004). The world wide web consortium (W3C) defines a web service as

"a software system designed to support interoperable software-to-software interaction over the Internet. It has an interface described in a machine-processable format (specifically Web Service Description Language (WSDL))",(Haas et al, , 2004). This shows that once a web service is up and running, any other system or application can request for the services given the right access. At this point it is clear that web services architecture are facilitated by three main standards i.e. WSDL, SOAP and UDDI.

Web Services operates on the idear of SOAs and it is a loosely coupled application that is developed using XML which facilitates deliver an application to users as a service for easy accessibility irrespective of your location and platform.

2.3.1 Web Service Technology

Web service uses standard protocols such as TCP/IP XML and HTTP for its operations and not object model specific protocols like DCOM,RMI or IIOP any lon. Web services L (**?**).

2.3.2 W3C Web Service Architecture

The world wide web is an internal organization that designs and develop Web standards, procedures and protocols that suports the world wide web. An example of these standards that the organization has set up is the Web Service Architecture (W3C WSA) which provides a framework to share a common definition of what a Web service is and the elements related to it. The figure below shows machine to machine interaction when ever a service is invoked over a network.

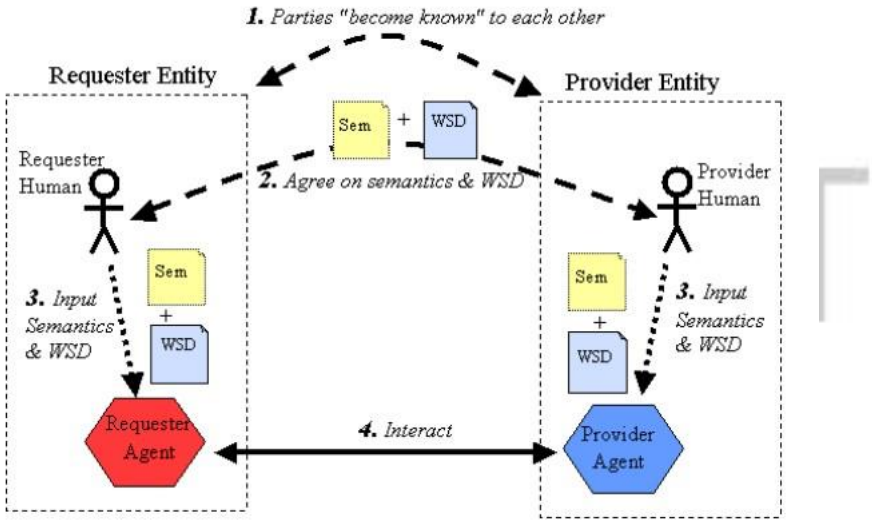

Figure 2.7: The General Process of Engaging a Web Service (Felipe , 2010)

1. A relashinship is established between the entity,requester and the provider even though sometimes it is only the provider entity that is known by means of service broker. 2. An agreement is also established which is also supported by the service registry which allows the provider to register a description of the service of which the requester examines to see if it fits his needs. 3. Agents receives WSD and semantic as input, to realise the service 4. Communication is started by means of exchanging messages between the requester and the provider agents that represents the entities. From the digram, three main elements are mentioned. These are: **Agent**:An agent is an application that acts as a requester or a provider and sends or receives messages during interactions. **WSD and semantic**: The WSD contains detailed information about the service such as message format, data types, protocols, service location and the interaction mechanics that can be expected when invoking the service. The semantic part ensures understanding of the concept and the behaviour between the requester and the provider. **Message**: A message is unit sent between agents. It is composed of a header, which contains metadata and it is defined in the WSD. It also contains a body which contains the message content.

2.3.3 Web Service Description Language (WSDL)

WSDL is a tool the (W3C) has recommended and it is defined as a virtual and concrete interface of a web service. (Leitner , 2007) (World Wide Web Consortium , 2002) WSDL is an XML language and that makes it platform independent.Within the WSDL file are the parameters that described the service. It shows what the web service renders, its location and how it can be invoked.(**?**) With WSDL, the virtual definitions are separated from the network deployment and binding details. One of the functions of the WSDL is its interface description language. This makes it similar to the IDL which is used in distributed object middleware. This similarities between the IDL and WSDL has problems and misconception. WSDL currently has two versions, which is the versions 1.1 and 2.0. Currently the lower version which is version 1.1 is the most widely used one compared to the 2.0. Even though the version 2.0 is the latext version, the problem with it is it is not widely supported. This has discouraged people from using it.(Leitner , 2007)

2.3.4 Structure of WSDL

WSDL version 1.1 has six major elements that makes up the WSDL descriptions. These elements are (Leitner , 2007) 1.Type: This provides the data type elements using XML schema 2.Messages:This represents a virtual notifications that service accepts or sends 3.Port types: These are abstract operations 4.Bindings: It binds the porttype to a specific protocol and data format specification 5.Ports This is normally considered as the end points of bindings that the address of a certain. 6.Services: These are sets of ports that they group a number of related ports. WSDL also has one unique advantage which is its language extensibility which it uses to provide language bindings for SOAP and HTTP.(World Wide Web Consortium , 2006) Belew is a WSDL

estensibility

```
<wsdl:service name="MessageBasedOrderService">
  <wsdl:port binding="impl:OrderServiceSoapBinding"
             name="OrderService">
    <wsdlsoap:address location="http://localhost/OrderService"/>
  </wsdl:port>
</wsdl:service>
```

Figure 2.8: WSDL extensibility

WSDL operates on the Message Exchange Pattern (MEP) which means that the WSDL endpoints are often invoked in a request /response manner. WSDL also supports the following in terms (World Wide Web Consortium , 2002) of communication:

- Single Direction - there is no message response after receiving it.
- Request-Acknowledgement - the endpoint receives a message and an acknowledgement is sent.
- Solicit-Acknowledgement - a message is sent and an Acknowledgement is received .
- Notification - a message is sent and there is noAcknowledgement .

WSDL Binding Styles WSDL is used together with SOAP in several occations to form a WSDLto-SOAP binding. This is made possible through two main predefined styles (Leitner , 2007) which are RPC style and document style. These two encoding styles have two main uses which are literal use or encoded use. Base on this we have four different combinations:RPC/encoded, RPC/literal, document/encoded and document/literal, all of them mutually incompatible. To rectify these anomalies, an organization known as the Web services interoperability organization (WS-I) was formed and they have released a basic profile that defines interoperable Web services. This basic profile that the organization released banned the encoded use because of its interoperability issues and recommended the use of document literal instead. Recent SOAP frameworks have adopted the

WS-I recommendations and have ignored the RPC/encoded style as an opted for the document/literal. The document/wrapped or document/literal is the most widely (Web service interoperability organization , 2014)used version because of its wrapped parameters. It also has further and better confinements:

- Messages in document/wrapped has one message parameter.
- The wrapper has a local name equal to the operation name of the operation that this message is associated with.
- The wrapper type is defined using the sequence compositor. Other compositors (all or choice) may not be used.
- The wrapper type has no attributes.

In practical terms, all wrapped/documents have one name which is the same as the name of the operation to be invoked. This is a merit beacause the operation name is contained in the SOAP as the name of the wrapper type. This makes it possible to use XML schema validator to validate the WSDL descriptions. The only demerit (Butek , 2014) of this style is that it can not support overloading of WSDL operations. A typical WSDL file or document is a machine and human readable language used to describe web services. It is made up of mainly XML grammar generated from a web service and it describes the methods and functions exposed in the web service. In a typical WSDL file, one may also find the data types of the method or function parameters, the adress which is the Uniform Resource Identifier(URI) and the protocol carrying the service. Its actual content is all the necessary information the user needs to invoke the service. Figure 2.2 and 2.3 shows a typical wsdl file generated from a prototype web service. Figure 2.2 depicts three sections of the wsdl. The first section is the header of file that describes XML version, message encode type

and the XML criteria resource. The second portion is the interface description of services object. In this example it shows the names of the methods (two of them) being exposed as services and their data types. Thirdly, the operations that the service will execute are encoded here. The location of service as i.e. "http://127.0.0.1:9876/ts" can also be seen in the last section of the wsdl file.

2.3.5 Work-flow of Web Services

Web services work on the principle that a Service provider may have to create the web service and its service definition and (UDDI) is used to publish it the service registries. As soon as the service is published or made known to the public any body who wants to consume the published service can do so through the UDDI.The service requster looks for the registry and the URI that points to the web service. The requsterIt binds these two information to its and its able to invoke the service

Figure 2.9: Work-flow of Web Services

2.3.6 Web Services Limitations

Transaction

There is no atomicity since HTTP is a stateless protocol mean while business processes and transactions are useful

Security

Security is an issue since there is the need for add-on measures like encryption to deal with the insecure Internet transportation.

Reliability

It is justified that all the transport protocols suchas as HTTP,FTP,SMPT cannot addresses issues concerning reliabilty ,safe delivery and elimination of duplicated transaction (Leitner , 2007).

2.3.7 Extensible Markup Language (XML)

XML has come a long way since its advent. XML can be used to structure electronic files or documents. By so doing the actual content is detached from the way its presented. Due to its ability to work with data, it is now seen in most kinds of data representation including databases (Pelz-Sharpe , 2010). XML marks different sections of a document with custom labels or tags which are unlimited. Because of its unlimited labels or tags a wider range of the document can be sectioned. XML is made up of markup tags which are wrapped around data in order to define it in a finer state. Data can be read easily from an XML based document by breaking down the marked sections or tags. XML documents are very simple to understand and move within other platforms. (Pelz-Sharpe , 2010). Due to its cross-platform nature, XML can be used to integrate systems written in different languages in a data driven style. In this case the systems would communicate by exchanging and processing XML-based documents.

2.3.8 XML and Documentation

As explained earlier, XML is a very powerful documentation tool. The advantage of XML for documentation is that it can be used to define the common traits in books, magazines, stories, advertisements, and so forth. The best thing about XML for documentation is that the XML is easy to understand by humans, both of the actual documentation and the XML code surrounding it(Kyrnin , 2014).

2.3.9 Universal Description, Discovery and Integration, (UDDI)

Instead of exposing a WSDL document to every client, the UDDI serves as a central hub which registers or hosts the WSDL document (services). By so doing the same service(s) will not be replicated to different clients thereby curbing redundancy. The main question still pends. What actually is UDDI and how does it function. UDDI was originated by the Organization for the Advancement of Structured Information Standards Consortium in year 2000. It is an XML-based registry which houses the data and meta data about the web service being published (Organization for the Advancement of Structured Information Standards , 2014). The information presented in a UDDI is written in xsd because it provides data in a more natural and hierarchical way. XSD is very rich when it comes to data types and also has the natural ability to validate information encoded in any type of schema.(Organization for the Advancement of Structured Information Standards , 2014). In size comparison, the UDDI registry is much larger than the WSDL document. This is due to the fact that the UDDI actually has the entire meta data of a particular service. For instance a service that displays the business details of a client will have all the attributes about the client like his or her first name, last name, address etc listed in the UDDI while the WSDL file will contain only the service name and other descriptions about the service.

To sum it up on UDDI registries, let us discuss about their flavors. UDDI registries exist in three basic types or flavors. An organization can actually choose any of the flavors based on the type of system being implemented or deployed. There are three main ways of implementing UDDI registries and this include Private also known as Corporate, Affiliated and Public. Figure 2.6 showcases the conceptual illustration of how the registries are affiliated or related.

2.3.10 Simple Object Access Protocol, SOAP

SOAP is an XML that serves as a link between the service requester and the web service. It actually defines a standard protocol for passing lightweight XML messages or information between distributed systems or applications (Du , 2004). SOAP also depends on HTTP(S) to send its messages . Because of its XML nature its is very much inter operable and can seamlessly send messages across systems with different operating system, hardware, network and applications of heterogeneous programming languages.Every SOAP message consists of an envolop, a header and a body. A simple code illustrating a SOAP envelope is shown in Figure 2.7.

The envelope makes the XML a SOAP message and that differentiates it from a normal XML. Without the envelope, the document would just be any XML document hence translated into another meaning. The SOAP Header tags contain any information the provider would want display.

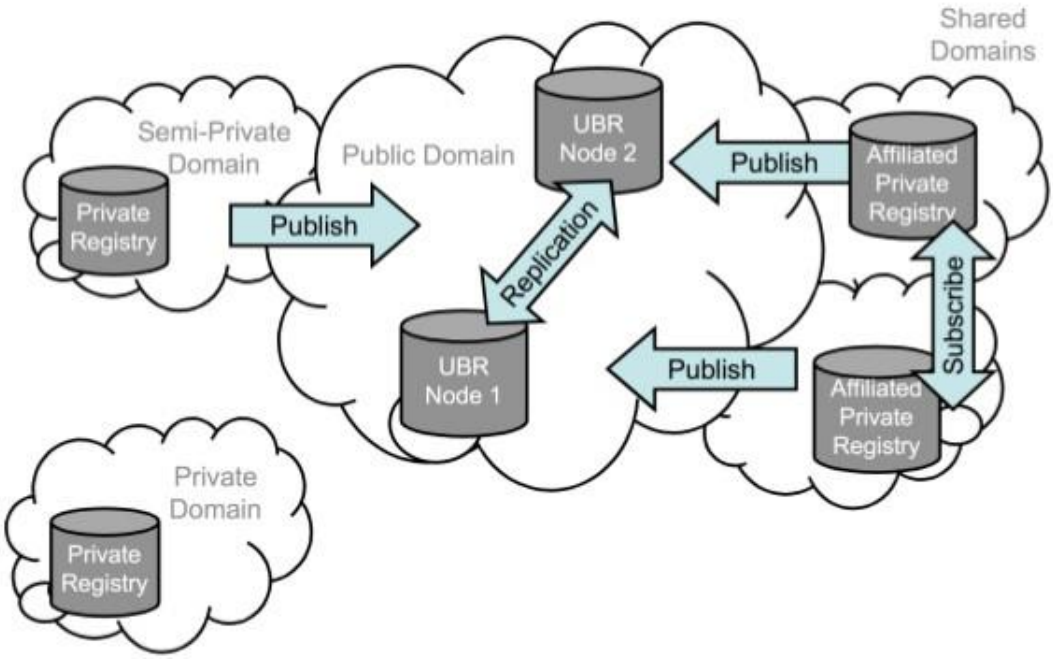

Figure 2.10: Types of Registries and their relationships (Organization for the Advancement of Structured Information Standards , 2014)

The header tag is not very much required. The main service information are placed in the SOAP Body tags. It contains all the calls and responses information. A typical example is presented in figure 2.8. Lastly in this chapter, we present an example of a SOAP architecture based on a Java Web service.

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV = http:"//schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle=http:"//schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Header>
//Header part
</SOAP-ENV:Header>

<SOAP-ENV:Body>
//Body part
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 2.11: A SOAP Envelope

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice xmlns:m="http://www.example.org/stock">
      <m:StockName>Lenovo X230</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Figure 2.12: A typical SOAP Envelope

In the above architecture, a client application (another programming language) makes a remote procedure call over HTTP using SOAP. SOAP runtime environment passes the request to Java object and waits for the response from it. Methods of Java object can access both local and remote resources then pass them through the SOAP runtime back to the client application.

2.3.11 Representational State Transfer (REST)

(REST) can be defined as a set of architectural constraints which are defined by fielding and are used (Felipe , 2010) implement RESTful services. These architectural constraints can be categorized into: a stateless client/server protocol; a uniform interface; use of hypermedia; a universal syntax for addressing; self-descriptive messages. The costraints facilitates the creation of services which are exposed to APIs. The created services can be consumed through the client/server protocol. This protocol allows all kinds of tools like desktop applications or web applications

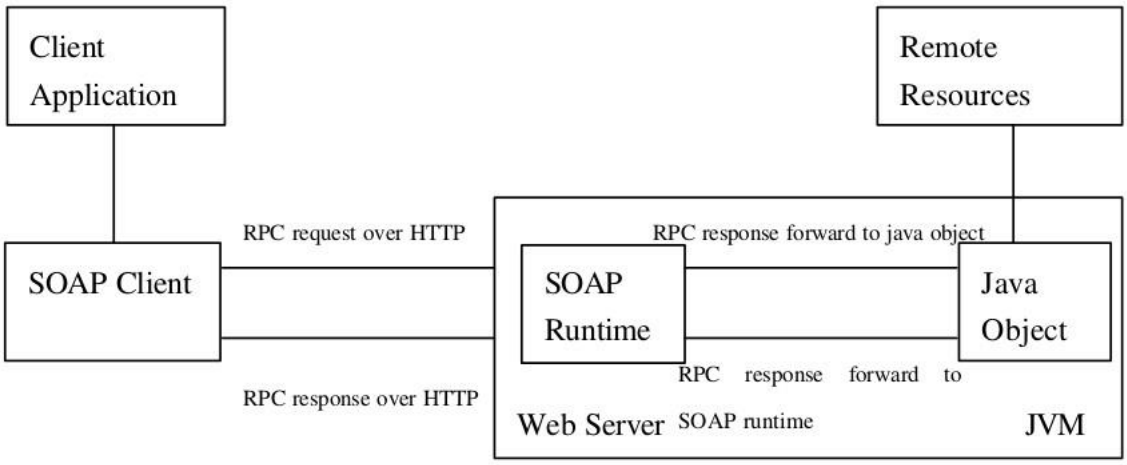

Figure 2.13: SOAP Architecture (Du , 2004)

to access or consume the service provided they have the ip address to access the service. The hypermedia and the uniform interface which are part of the architectural constraints of REST allows clients to surf for services and consume them without any need for the codes from the service provider. REST architecture is stateless and that makes it more scalable. This charateristic ensures that the server does not keep any previous information about clients. The selef descriptive messages helps to decouple resources from their representations so different media types can be used to display its content. The minimal information unit in REST is a resource. Resources are data sources which keeps the functionalities and the application state of a system.

All resources are to have a unique names so that they can easily be identified. The identification is done through some sort of globaly unique identifier(GUID). A resource can not be accessed directly but it can be manupilated by means of representations. Through representations a client is able to access a resource, manipulate and transfer it to other components. The world wide web is known to be implemented on REST and this is used to explain how the world wide web works on HTTP. REST uses HTTP as an application layer and for that matter uses its mechanisms, metadata and semantic elements to provide RESTful services by means of implementing

its contraints.

The uniform interface in such implementation is obtained through HTTP methods: GET, POST, PUT and DELETE, which defines a CRUD (Create, Retrieve, Update, Delete) interface for any REST resource. HTTP also provides stateless interactions through hyperlinks. Resources are identified by means of URIs (Felipe , 2010) which provide with a syntax to build unique identifiers. Finally, self-descriptive messages are obtained by means of MIME types which allows decoupling resources from its representations through different formats (e.g.: HTML, XML, JPEG). These formats are open standards, so any client can understand them.

2.3.12 SOAP vs. REST services

Below is a detailed comparison between REST and SOAP From ealier reviews it was made clear

	REST	WS-*
Uniform interface	Yes	No
Transfer protocol	HTTP only; used as application protocol	TCP, FTP, HTTP...; used to transport SOAP envelopes (often considered an abuse)
Payload format	XML, RSS, JSON, YAML, MIME	XML (SOAP)
Service description (WSD)	WADL[19] has recently become a standard, but it has not earned popularity. Common approach uses human-readable documentation, although last version of WSDL can describe REST services.	WSDL
Service discovery	It is not strictly necessary	UDDI

Figure 2.14

that REST has certain charateristics that provides several advantages over SOAP. Rest:

- REST requires a minimum infrustructure support.

Service composition	Mashups, do-it-yourself	WS-BPEL, do-it-yourself
Service identification	URI	URI, WS-Addressing

Figure 2.15: Web service technologies comparison.(Felipe , 2010)

- REST is scalble, simple and low performance overhead
- It has a uniform interface that do not give problems to APIs and this has given a high rise of services. REST applications are executed in the web browser using the HTTP protocol.
- It creates a uniform platform for all resouces and it does not require WSDL to define it.

2.4 Web Service Security

Web applications, web service and the Internet as a whole cannot be fully secured hence the HTTP being vulnerable. Eaves droppers can listen on an HTTP in other to siphon pertinent information or messages. In web service development, it is very imperative to incorporate some security features so as to protect the data being sent. In fact there are several ways that security could be improvised into web services. A survey conducted by Web Services-Interoperability Organization (WS-I) identified some key threats opposing Web services (Government of Hong Kong , 2008). These threats includes the threats associated with http (the man-in-the-middle-attack). Their proposed solution was to find a means to secure the transportation channel using https.

2.5 Development Tools and Technologies

In developing the web service certain tools and technologies were paramount. In order to gain ample experience and not reinventing the wheel, certain third party products and software were

adopted. The first which is discussed here is the renowned eclipse integrated development environment (eclipse IDE). The eclipse IDE has an extensible plug-in system for customizing the development environment. Eclipse IDE which support alot of application development tools like Ruby ,Python, PHP, C++ etc based on its plug-in capabilities. It is licensed under the Eclipse Public License which is free. The reason why eclipse IDE was chosen was the fact that it supports a whole of plug-ins for the deployment of a web services. This makes it very easy to design and deploy web services in such an environment. Due to these capabilities and some other favorable factors, it was chosen over other IDE's for this study. Another essential technology or framework when it comes to the deployment of web services is the Apache web server.

The programming language used in developing the proposed system is the widely used scripting tool, Hypertext Preprocessor version five(PHP 5.0+)which is a scripting language was used because it suports the SOAP connection as a client and as a server. With the versions 4.0 and below a pseudo class library called NuSOAP needed to be included. As already mention PHP can be used to develop a web service and a corresponding client to consume it. Since the demonstration of the cross-platform capability of a web service is keen in the writing of this thesis, the PHP is used as the web service server while other languages like Java and Python were used as the clients.

2.5.1 PostgreSql Database

The database management system chosen from this study is PostgreSql. PostgreSql is worlds most advanced object-relational database management system. It is free and open-source software. It is developed by PostgreSql Global Development Group consisting of handful of volunteers employed and supervised by companies such as Red Hat and EnterpriseDB (Andurkar , 2012). MySQL and PostgreSql both compete strongly in field of relational databases since they both have advanced

functionalities and also comparable performance and speed and most importantly they are opensource. PostgreSql is Object-relational in the sense that it is similar to relational database but its database model is object-oriented. Objects, classes and inheritance are directly supported in database schema and query language.(postgresql.org , 2014).

2.6 Service Developmental Strategies

In the chapter 3, a simple prototype web services that displays the current date and time on a remote server is developed. This is to enlighten readers on how an actual web service would function. For any chosen programming language, there will always be a service interface, a service implementation class, a publisher and lastly a remote client which can be of any other programming language. In this work, the client service will reside on the banks' servers where as the main service will be hosted on the institutions' servers. The transaction is expected to be in real time. It may be some few seconds. The service is developed using PHP programming language. As already stated, it does not necessarily mean the client should also be developed using the same language. The implementation of web services ensure easy integration between heterogeneous systems since it is endowed with a flexible communication port which improvises XML and HTTP to be precise. The following sections describe clearly how the proposed web service is developed.

In developing a web service, one can actual stick to two main styles or processes. There is the code first approach and the contract first approach. The code first approach is sometimes called the bottom-up approach. In this approach we first start with the source code of the web service and then eventually publish it as a service i.e. developing the main service first. More concretely, the code first approach helps to easily convert an existing application into a web service. On the other hand, the contract first approach is very much analogous to the code first approach. It is most often

referred to as the top-down approach. In this style, we first write the Web Service Description Language (WSDL) document according to the service contract. Entities participating in the service invocation come up with a set of APIs (methods and functions) and then map them into the WSDL document. Once we have the WSDL document, we can now go ahead to write the code and then complete the business logic of the main service. In both cases the web service would function as expected but notably, with the code first approach, you might not be able to harness the full power of the service, but you can still achieve your goal. In this research work, the contract first approach was used. The following sections describe both approaches in details.

2.6.1 Code First Approach

This approach is the same as the one described when developing the test case. In this approach, the web service methods are developed first. Here our main concern is to develop the functionalities of the web service and then use a third party means to auto generate the contract document which is the WSDL file. Though web services developed with this approach works perfectly, the developer is not given full power to the generation of the contract document. This makes it difficult when more capabilities are supposed to be included in the contract file. To sum it up, the approach says echoes that services is to be granted first before a contract which is contrary to real world situations.

2.6.2 Contract First Approach: WSDL

In real world before a person or an entity provides a service to another entity, there should be a form of a written contract between both entities. In this contract all the business logic of the service is written in order for the service provider and the client to have a guide through the transaction.

In the same manner, creating web services with the contract first process deals with the contractual terms of the service first. This approach starts by developing or writing the XML Schema/WSDL document or contract first followed by the code for its implementation. The point to note here is that once the client and the provider of the service have the WSDL document, it acts as a contract according to how the service is to be developed. The contract document describes the format of a request and response, the service endpoint location and other useful things like security implementations. Before we start writing the contract for our Data Communication Service lets discuss some types of contracts there is. There are two main types of contracts in the world of web services. We have the data contract and the service contract. Both of these two contracts are fused together to form the WSDL document. In the data contract, we define the message formats that our service will accept. The most common approach used in creating the data contract is the use of XSD which is an XML Schema. In creating the data contract for the Payment Service, the data types of the the parameters of the methods which will be exposed as well as their return types are written in the XSD file and saved with a **.xsd** extension. An example of this file is included in the appendix for reference. The second part is the service contract. The service contract file is more or less the WSDL file. It is basically an extension of the data contract file. In writing this file we start with the WSDL definitions and include the target name space of service. Next we have to include the data contract in the WSDL element labeled types. Table 3.1 shows the essential portions of the remainder of the service contract file.

Table 2.1: Elements of Service Contract (w3schools , 2014)

Section/Element	Description

< message >	Defines the type of data being communicated in the activity
< portType >	Specifies all the operations which are supported by the service endpoint
< binding >	A protocol assigned to a particular port type
< service >	Houses the service name and the location address of the service

The next on the deck is the inclusion of the WSDL port type element. The port type element houses the all the operations in the web service. It defines each operation in an operation element listing its input(request) and output (response) messages. So far we have discussed the abstract portions of the service contract file. What is left to complete this contract is the binding and service elements. This constitutes the concrete part of the service contract. The binding element communicates with the client on how to invoke or call the operations in the port type element.

2.7 Summary

Most of the the works done in the area of web service talks little about the reliabilty of web services, performance and security. It is necessary and important to look at these areas which have received little attention by researchers. literature governing this work speeaks less about web service security and performance. We proposed a framework to develop a web service and measure its performance as well as incorporate security mechanisms in it to ensure safe delivery of data.

During the study we proposed a secured way of sending data through heterogeneous applications and a systematic analysis of the performance of the service insatnces that have been developed.

Chapter 3

Methodology

3.1 Introduction

This chapter of the thesis dwells on the deployment and development of the web service. The existing system at Kumasi Polytechnic and its ”modus operandi” was studied. In studying the existing system’s operational mode, another means was suggested to effectively beef-up the security of the service. During the development a simple test case was deployed to give a fair idea about how the main service would function. Various technologies and tools needed for the development were also discussed. In every system, performance measurement is very keen to ensure optimum usage of the system. In order to determine the performance of the web service developed, some key properties such as the load exerted on it, the bytes of data it processes and the time it takes for the processing is measured. This was done to ensure efficiency of the service.

3.2 Test Case Development

This section of the thesis deals with a hypothetical example of a web service and its client for invocation. In other to pose a mock up description of web services two simple applications that would consume a given Java web service were developed using Java and PHP programming languages. The PHP was used in other to establish the heterogeneous capabilities of web services. This simple exercise was to demonstrate how a Java web service is created, published and invoked

by several clients. In this section the idea of homogeneity/uniformity and heterogeneity/non uniformity in programming languages were demonstrated through the service clients. The main simple duty of the web service developed in this section was to print the current date and time of a server. The simple applications which are Java and PHP made a remote call to the service, invoked it and then the current date and time on the server was displayed. To start it all, the service object or interface was developed. The service interface is the part of the web service that defines all the methods or functions that will be published for public consumption. In actual fact, it is the warehouse of the web services. It has all the methods needed by the client to consume. Whenever the client makes any call or invocation, it will actually be searching for the the methods declared in the services interface. Figure 3.1 shows the simple code describing this service interface.

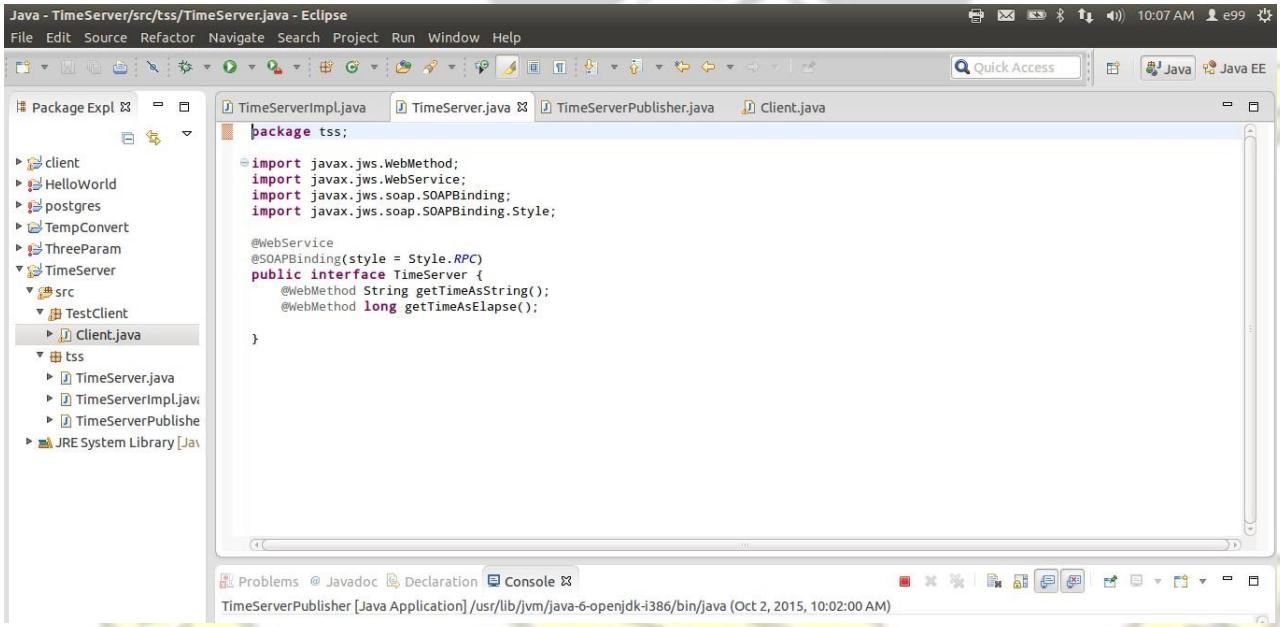

Figure 3.1: Service Interface

The next thing to do was to write the service implementation object or class. The service implementation class actually defines what the interface method does. In this test case, it defined the codes that retrieved a systems current date and time. This class extends the service interface in order to use the method declarations in the interface. In this class, also the data types used to

declare the methods in the interface was maintained for consistency. A screen shot of the service implementation class is shown in figure 3.2.

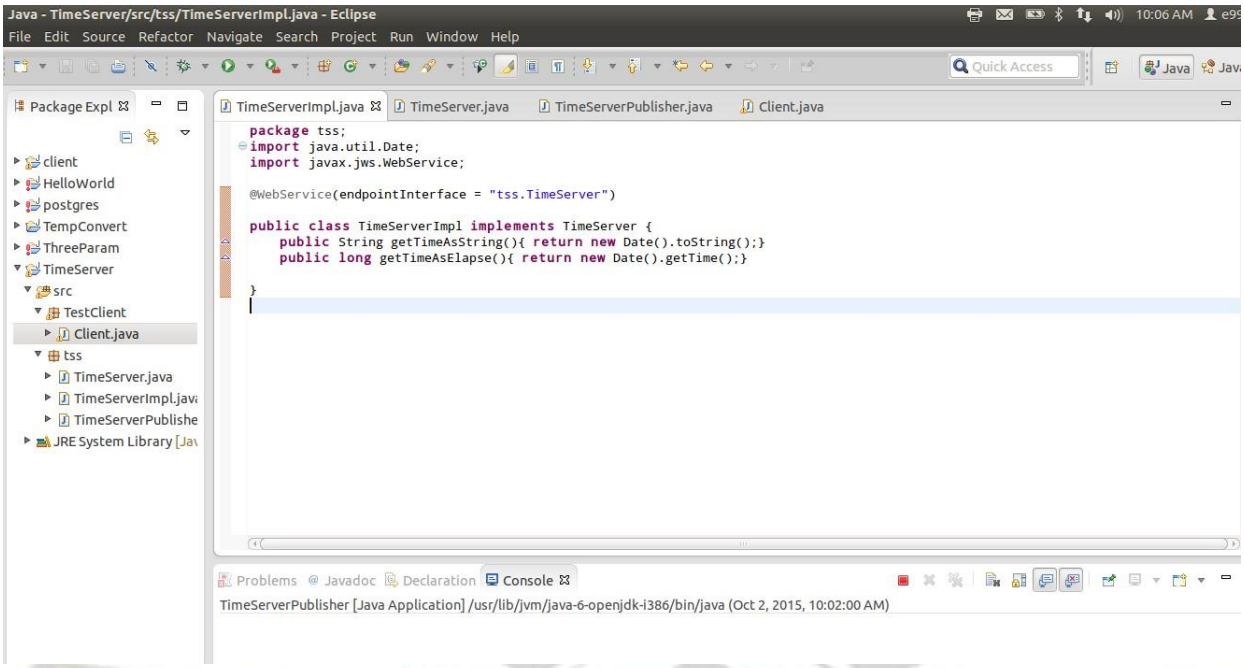

Figure 3.2: Service Implementation Class

After the implementation class, comes the publisher class. This class defined the service endpoint i.e. the URI where the service was deployed in a Web Service Description Language (WSDL) file. When the publisher class was executed, it auto-generated the WSDL file which sets the service up for client consumption. A sample code of this class is found in figure 3.3 and as well as the WSDL in figure 3.4.

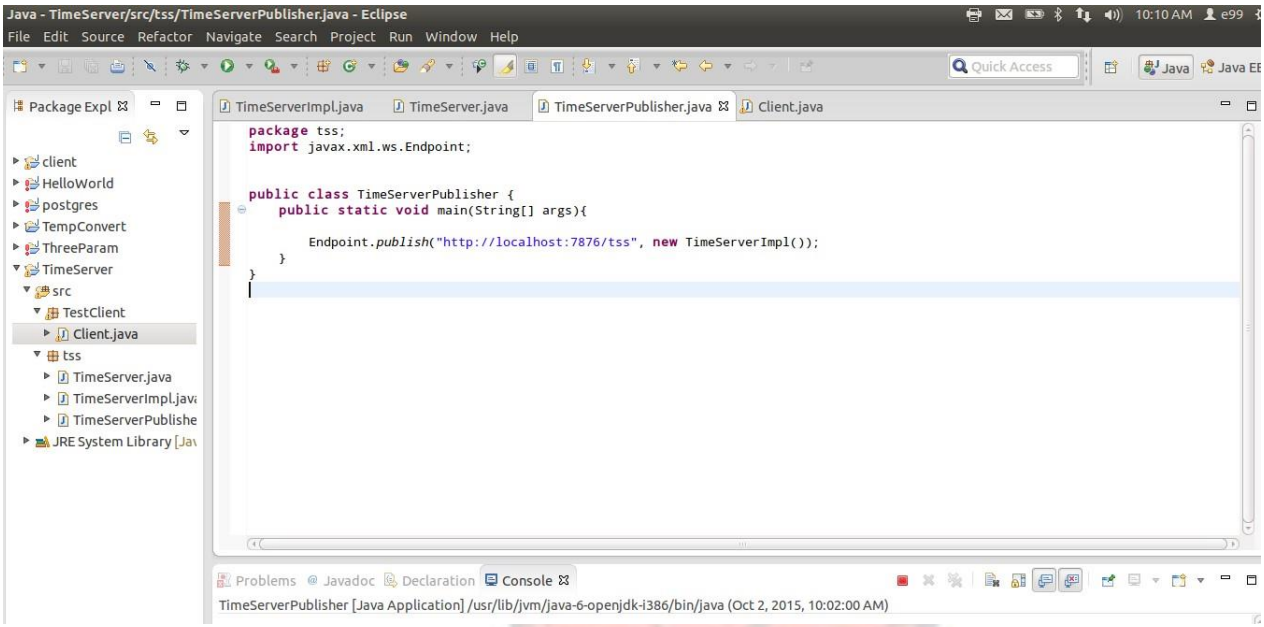

Figure 3.3: Service Publisher Class

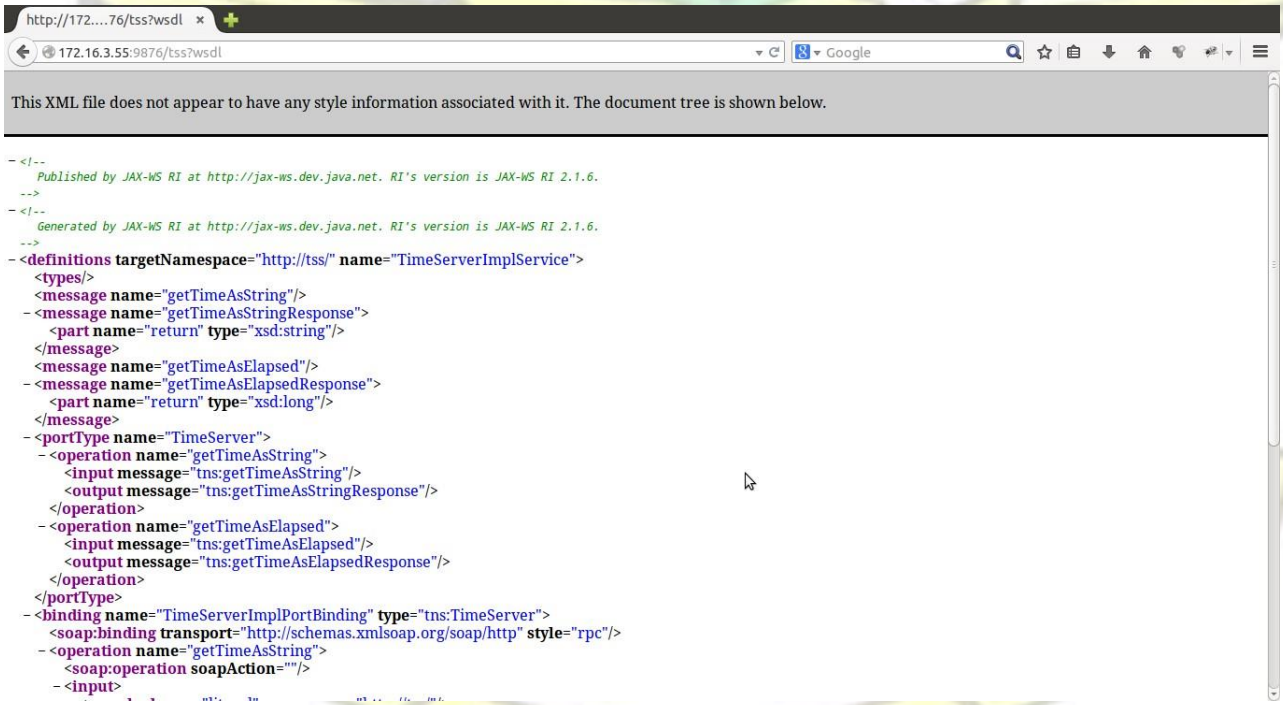

Figure 3.4: Auto Generated WSDL File

3.2.1 Deploying over Java Client

When the Java client code in figure 3.5 was executed, a connection was made with the WSDL file at the specified URI through HTTP. The client then searched through the WSDL file for the < *portType* > tags. Within this tags are sub tags specified as < *operation* >. The client then retrieved the value of the name attribute which was the web service method or function and executed it. Sequel to this action, a client response which was the date/time of the remote server was generated as shown in figure 3.6.

```
package TestClient;
import Client.TimeServer;
import Client.TimeServerImplService;
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
import javax.swing.JOptionPane;


import java.net.URL;

class Client {
    public static void main(String args[ ]) throws Exception {
        URL url = new URL("http://172.16.3.55:9876/tss?wsdl");

        QName qname = new QName("http://tss/","TimeServerImplService");

        Service service = Service.create(url,qname);

        TimeServer eif = service.getPort(TimeServer.class);

        JOptionPane.showMessageDialog(null,eif.getTimeAsString());
        System.out.println(eif.getTimeAsElapsed());
    }
}
```

Figure 3.5: Java Client Code

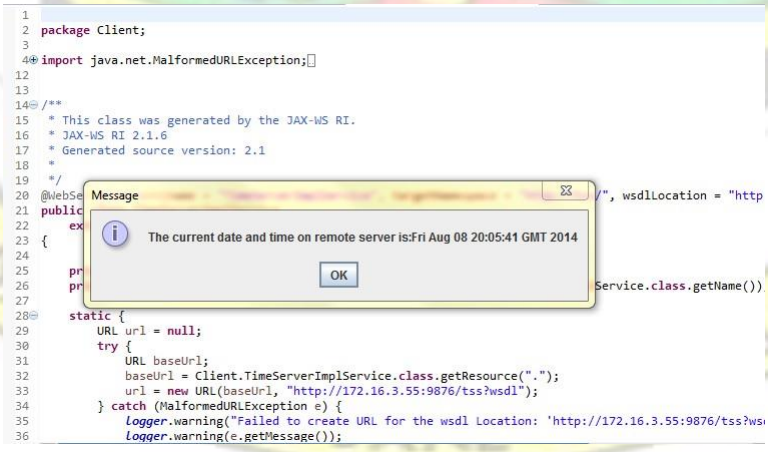

Figure 3.6: Java Client Response

3.2.2 Deploying over PHP Client

Aside the deployment in the Java based environment, the service instance was also tested using a PHP Client. XAMPP (Apache, MySQL, PHP and Perl) web server was setup on a Microsoft windows operating system. Cascading Style Sheet (CSS) was used to style the PHP client in order to give it a flashy look. The web service was invoked at the url: **http://localhost/xampp/realapp/test.php** through a browser. The PHP client was also able to retrieve the methods there by displaying the current date and time on the remote server. This is shown in figure 3.7.

Figure 3.7: PHP Client Response

3.3 Existing System

For the past years, the Kumasi Polytechnic Institute have had many challenges with payment of student tuition fees. The system operated by the institution is not real-time since fees paid by students at various banks takes a day or two before it is reconciled. Normally the fees are sent in a spread sheet format with most of its fields distorted. Personnels at the institutions' finance office have to always spend weeks before getting the spread sheets into a proper order. The whole process is so manual that it is prone to a lot of human errors such as misspelling student names and

sometimes misquoting the amount paid by the students. In some cases the amount paid by the students are over estimated and/or under estimated. In solving this problem a real time system was developed to eliminate much of the human errors.

The developed system was to seamlessly interface between the banks and the schools' databases for data communication. In choosing the appropriate technology for the system, much consideration was given to the systems run by the financial agents (banks) of the school. A situation whereby the agents would be made to entirely change their current software was avoided. Web Services are able to seamlessly integrate heterogeneous systems to share common data. In the literature review of this thesis much information has been given on web services.

The existing system utilizes plain HTTP in accordance with XML to perform the data transaction. The security level implemented is just a password authentication on the server side. By this structure one can only send data if and only if his password exists in the schools database. The whole process begins with a a connection request from the client to check whether there exist a connection. If a connection exists, the service will then be allowed to post data into the schools database through the soap envelop path so long as the client submits a correct password alongside the payment details.

3.4 Proposed System

The proposed system was designed in other to eliminate security loop holes. Situations where by the message will be intercepted, changed and resent into the schools database was critically looked at (man-in-middle-attack). In such a case a sniffer would be sniffing on the network, retrieve the sent message, alter the fees with wrong values and resend it to the schools database. This loop hole was determined using the tcpdump utility on linux to capture raw packets of data in transit from the existing system. Through this process, all the essential information transported to the system

were seen. This can be seen at section 4.3.4.1 in chapter 4. In view of this, a single key data encryption method was used to encrypt the data before transmitting it over a secured channel (SSL) to remedy the situation.

3.4.1 Message Ciphering

A single key encryption algorithm base on modulo arithmetic as illustrated in section 3.4.1.1 was used to cipher the message string using python programming language. Each character that constituted the message string was converted into an integer using its position in a given alphabet of characters. A key value which is an integer was then added to the position integer and the result fed into a modulo function. This produced a new position integer for each character. The new position integer was then fed into a number base function. The number base function transformed all the position integers into a large integer before it was sent. This part of the process occurred at the encryption stage of figure 3.8. In this case, a network sniffer would see these integers instead of the original transaction. Using this technique, the sniffer can still get access to this large integer and temper with it. The question is, how can the sniffer be prevented from intercepting the message. To remedy this, a checksum algorithm was implemented and the result was padded alongside the large integer. The large integer was again fed into another modulo function and the anticipated result which was to be a two digit integer was padded at the extreme right of the large

integer.

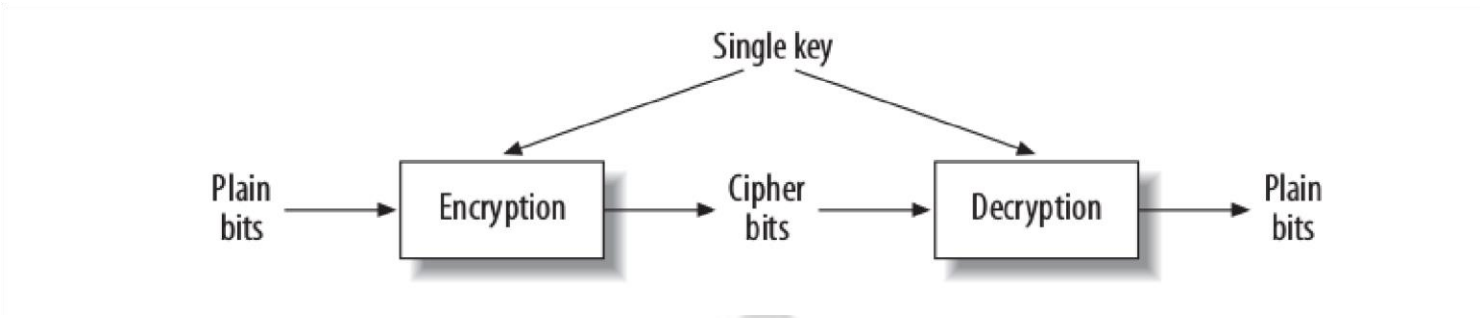

Figure 3.8: Single Key Encryption

3.4.1.1 Ciphering Algorithm

In this section the single key encryption algorithm that was designed for the ciphering of the raw transaction has been presented.

1. input Original Message, [OM_i], *key*,*k* where i is the index of each character in the message including spaces and k is the encryption key
2. define Character Set, [CS_i]
3. for each $om_i \in [OM_i]$
4. if $om_i \in [CS_i]$ then $i = om_i \in [CS_i]$ and the new index, $j = (i + key) \bmod M$ where *M* is the size of [CS_i]
5. Code Index, *ci* = [*j*]
6. set Message Integer, *MI* = 0
7. for $j \in [ci_j],\ MI = \sum_{j=0}^{size(ci)} Z^j(ci_j)$ where *Z* is a double digit integer
8. new input message string, *IM* = *MI* concatenated with *MI* mod *N*

3.4.2 Message Deciphering

The next level was to decipher the transaction on reaching the server side which is endowed with the decryption algorithm described in section 3.4.2.1. This stage of the process happened at the decryption side of figure 3.8. As the server gets the large integer, it strips off the last two digits which is the checksum. The server then computed a modulo arithmetic of the large integer and compared it to the checksum. If these numbers are similar then the transaction was not tempered

with. If they are not similar then the transaction must have been altered hence the server will reject the transaction and notify the client. In an event where by the checksum were the same, a reverse of the ciphering mechanism was computed over the large integer hence the original transaction was retrieved and inserted into the database. Another means of ensuring double level security was to use Secure Socket Layer (SSL) which is described in much detail in section 3.6 of this chapter.

3.4.2.1 Deciphering Algorithm

This section presents the algorithm for the decrypting portion of the implementation.

1. accept input message string, *IM*
2. strip off the checksum as pd
3. if $pd = MI \bmod N$

   while $MI \geq Z$ $MI =$

   MI/Z

   input message string, $MI = [MI \bmod Z]$ end

   for i in $[MI_i]$

   $new_i = (i + 36 - key) \bmod M$ Decoded

   Message + = $[CS_{new_i}]$

   else break end

3.5 Web Service Performance Test

In testing for the performance of the web service, the following questions were set as a guide through out the testing.

- Does the service respond with the correct values?
- How many transaction can the service send withing a given time?
- Can the service handle expected and unexpected user loads?

3.5.1 Test Suite:soapUI

SoapUI is one of the common tools or simulators used for the functional test of a web services. It is not limited to web services, though it is the de-facto tool used in web services testing. In web services testing, soapUI is capable of performing the role of both client and service. It enables users to create functional tests quickly and in an efficient manner by using a single environment.

3.5.2 Functional Testing

Web Services with incorrect responses can lead to problems. Web Service Functional Testing ensures that the web service is functionally correct. This was done by parsing the alphanumeric value **yaw nti 05120000314 1 tsn64** directly to the service to test whether it would respond with the expected output. The result is shown in figure 3.9. The figure shows the soap faults or errors which was generated due to wrong parameter sets. The soap fault reads **postgres7 error:**

[-1: ERROR: ValueError: invalid literal for int() with base 10: 'yaw nti 05120000314 1 tsn64'.

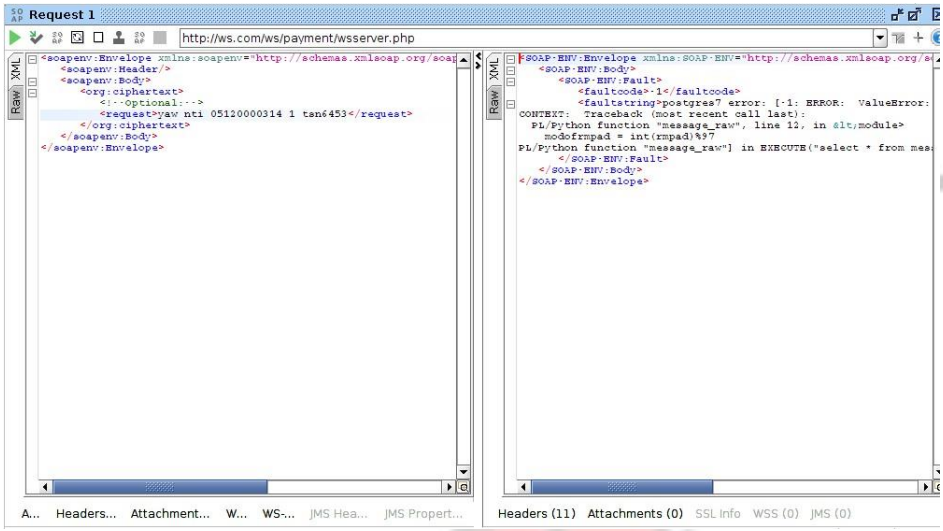

Figure 3.9: Soap Fault Generated by soapUI (Error)

3.5.3 Performance Testing

Once the functionalities of the web service had been satisfied, another test that was carried out was the performance test. This stage of service testing is most often called non functional testing. Here the testing was based on number of transactions per minute, the load on the service and the size of message in bytes.

3.5.3.1 Number of Transactions

During the simulations the following parameters were tested in other to have a clearer view of how the service performed. These parameters included load test, number of transactions per a given time and bytes of data transfered. In testing for the number of transactions the web service might send with a given time, it was invoked for 60 seconds. The execution was performed for all the three services on the local development server. Each web service was invoked 10 times in other to obtain the mean number of transactions.

Table 3.1 shows the transaction per minute for the unencrypted message over HTTP. This particular service was invoked 10 times and each lasted for 60 seconds. In the table the 3rd trial of the experiment registered 409 transaction per minute where as the 10th trial had 402 transaction per minute. Averagely the web service was able to send 403 transactions per minute.

Experiment	1	2	3	4	5	6	7	8	9	10
Number of Transaction	398	408	409	402	401	403	403	402	403	402

Table 3.1: HTTP with Raw Message

The same experiment or investigations were repeated for the encrypted message or transaction with both the HTTP and HTTPS transportation media. Invoking the client at 60 seconds, both services generated the number of transactions as shown in the Table 3.2 in page 65. It is very clear that the HTTP medium allowed more transactions than that of the HTTPS transport medium. During trial 5, the HTTP registered 403 transactions where as the HTTPS recorded 383 transactions. The difference in transaction was about 20 transactions. The highest number of transactions in both cases are 406 and 388 respectively. On the part of the HTTP medium, the 406 transaction occurred during the first trial while as the 388 occurred during the 3rd, 8th and 9th trials.

Experiment Number	Encrypted HTTP with Message	with Encrypted HTTPS Message
1	406	383
2	404	383
3	403	388
4	405	381
5	403	383
6	402	382
7	403	383
8	404	388
9	402	388

10	403	387

Table 3.2: HTTP and HTTPS with Encrypted Message

3.5.3.2 Transaction Size

The simulations were carried out the second time to investigate whether the transaction size affects the number of transactions per minute. This experiment was performed by starting with a transaction size of one character which is equivalent to 1 byte. Gradually, the transaction size was increased from 1 byte to about 40 bytes. In the first set of the investigations, the same character string was used and varied from 1 byte to 40 byte. Secondly the characters used were mixed since that represented the ideal case of the usage of the web service. These two kinds of investigations were performed for the encoded transaction on both the HTTP and HTTPS media. The results of the two experiments are presented in the tables that follows. In Table 3.3 are the results for the same character HTTP transfer medium. Sending 2 bytes of data through HTTP is averagely equivalent to 30 bytes of data.

(Same Size(bytes) Characters)	Number of Transac-tions
1	403
2	406
3	405
5	406
10	407
20	405

30	406
40	408

Table 3.3: Transactions per Minute for Same Character (HTTP)

In table 3.3, 406 transactions were sent for 30 bytes of data. Similarly 406 transactions were equally sent for 2 bytes of data. The same results were obtain for 3 bytes and 20 bytes of data which registered 405 transactions. The highest number of transactions were obtained when 40 bytes of data were sent. It registering an average of 408 transactions.

During the mixed characters experiments, the number of transactions were found to be averagely the same. In table 3.4 are the values obtained for the mixed characters HTTP encrypted transactions. From the table 30 bytes of mixed characters registered 408 transactions which was the highest. It was rather amazing that the starting character registered an average transaction of 402 which was the least. Here 10 bytes and 30 bytes of mixed characters registered 406 number of transactions.

Size(bytes) (Mixed Characters)	Number of Transac-tions
1	402
2	404
3	403

5	405
10	406
20	408
30	406
40	407

Table 3.4: Transactions per Minute for Mixed Characters (HTTP)

In Table 3.5 the results for the same character HTTPS transfer medium are presented. Here sending 2 bytes of data averagely transacts 386 records per minute. In the same table, 387 transactions were sent for 30 bytes of data. The same results were obtain for 5 bytes and 40 bytes of data which. The highest number of transactions obtained was the 387 where as the least was 385.

Transaction Size (bytes)	1	2	3	5	10	20	30	40
Number of Transactions	386	386	385	387	386	385	387	387

Table 3.5: Transactions per Minute for Same Characters-HTTPS medium

During the mixed characters experiments in the HTTPS, the number of transactions were also found to be averagely the same. In table 3.6 are the values obtained for the mixed characters HTTPS encrypted transactions.

Transaction Size (bytes)	Number of Transactions
1	386
2	387
3	385
5	386
10	385
20	387
30	386
40	385

Table 3.6: Transactions per Minute for Mixed Characters (HTTPS)

From the table 3.6, 2 and 20 bytes of mixed characters registered 387 transactions which was the highest. The lowest number of transactions which was 385 were registered by 3, 10 and 40 bytes of mixed characters. Averagely the transactions were found to be between 385-387.

3.5.3.3 Load Test

The next experiment that was performed on the web service was the load test. The web service was subjected to several loads to determine its robustness and also to investigate whether the service would break if it is subjected to heavy loads. Five threads were configured to send one transaction each to the web service simultaneously. The experiment was repeated 10 times for each of the developed services. The results are presented in table 3.7.

Trial Number	1	2	3	4	5	6	7	8	9	10
Number of Transactions	912	910	909	915	913	918	917	914	910	912

Table 3.7: Thread Load for HTTP transmitting Raw Message

During the load test for the Raw message, the least number of transactions recorded was 909 where as the highest was 918. Both was registered during the 3rd and 6th trials respectively. Both the 1st and 10th trials also recorded the same transactions which was 912.

In the case of the encrypted message, the experiment was performed for both the HTTP and the

HTTPS transport media. The experiment was also carried out with 10 trials for each situation. The results have been presented in table 3.8. In the table, it was very obvious that the HTTP medium registered a little more transactions than the HTTPS medium. The highest transaction which was realized in the HTTP was 944 whiles that of the HTTPS was 927. Both values occurred in the 1st and 5th trials respectively. The transactions recorded in the HTTPS were between 910 and 927. That of the HTTP were between 935 and 944. The least number of transactions were 910 and 935 for the HTTPS and the HTTP media respectively.

Trial Number	Encrypted HTTP with Message	with Encrypted HTTPS Message
1	944	910
2	938	913
3	935	921
4	940	917
5	939	927
6	936	914
7	941	919
8	940	920
9	941	918

10	938	915

Table 3.8: Thread Load Results for Encrypted Transactions over HTTP and HTTPS

3.5.4 Testing Over a Network (LAN)

After testing the performance of the web service of the local server, another client was developed on a separate machine on a local area network (LAN). The web service was then invoked from that machine in other to determine the number of transaction within a minute. The experiment was performed for the HTTP with Encrypted Message and the HTTPS with Encrypted Message. The investigations were also carried out with 10 trials for each situation. The results have been presented in table 3.9. In table 3.9, it was very obvious that the HTTP medium continued to register a little more transactions than the HTTPS medium. The highest transaction which was realized in the HTTPS was 359

Experiment Number	HTTP with Encrypted Message	HTTPS with Encrypted Message
1	401	353
2	395	358
3	396	358
4	398	359
5	400	357
6	396	358
7	399	357
8	391	354
9	390	358
10	393	359

Table 3.9: Number of Transactions per Minute on LAN

whiles that of the HTTP was 401. In the HTTP, the value occurred during the 1st but it highest value occurring in the HTTPS were recorded at during the 4th and 10th trials. The transactions recorded in the HTTPS were between 353 and 359. That of the HTTP were between 391 and 401. The least number of transactions were 353 and 401 for the HTTPS and the HTTP media respectively.

3.6 Security Implementation

As discussed in chapter two, the security mechanism implemented in the payment service is the Transport Layer Security. This is because of the web services are based on HTTP transactions hence the need to use some of its security measures to secure the payment service. The kind used here is the Secure Socket Layer (SSL).

3.6.1 Secure Socket Layer (SSL) Implementation

The SSL transaction has two phases: the SSL Handshake (the key exchange) and the SSL data transfer. These phases worked together to secure the transaction. The handshake begun when the client connected to the SSL-enabled server, a secured connection was requested, and a list of certificates were presented. From this list, the server picked the strongest cipher and a hash function that it also supports and notified the client of the decision. Additionally, the server sends back its identification in the form of a digital certificate. This certificate usually contained the server name, the trusted certificate authority (CA), and the servers public encryption key. The client then verified that the certificate was valid and that the Certificate Authority (CA) listed in the clients list of trusted CAs issued it was valid. Upon verifying that the certificate is valid, the client then generated a master secret key, encrypted it with the servers public key, and sent the result to the

server. When the server received the master secret key, it then decrypted it with its private key. Here only the server can decrypt it using its private key. Both the client and server then converted the master secret key to a set of symmetric keys called a keyring or the session keys. These symmetric keys are common keys that the server and browser used to encrypt and decrypt the data. This is the one fact that makes the keys hidden from third parties, since only the server and the client have access to the private keys. This concluded the handshake and begun the secured connection allowing the bulk data transfer, which is encrypted and decrypted with the keys until the connection closes. If any one of the above steps fails, the SSL handshake fails, and the connection is not created.Though the authentication and encryption process may seem rather involving, it happens in less than a second. Generally, the user does not even know it is taking place. However, the user is able to tell when the secured tunnel has been established since most SSL-enabled web browsers display a small closed lock at the bottom (or top) of their screen when the connection is secured.

3.6.2 RSA algorithm for the Web Service Certificate Generation

A keystore was generated on the web service server. In this keystore was a self-signed certificate generated using the RSA algorithm. RSA stands for Ron Rivest, Adi Shamir, and Leonard Adleman the three authors of that algorithm in 1977. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. Messages encrypted by RSA can only be decrypted by the private key and only the client and the server knows that key. In RSA, party A carefully selects two numbers, M and e publicly. Party B will then use these two numbers to encrypt a message and then send it back to party A. During the transmission of the encrypted message from B to A, party C intentionally intercepts the encrypted message, M and e

since they are public (T. Davis , 2003). The duty of RSA is to make it practically impossible for party C to decrypt the message. This is because there exist a private key, d which is used to decrypt the message and it is only known by A. At the heart of the RSA algorithm is the RSA modulus, M. This modulus is chosen carefully by multiplying two distinct prime numbers, r and s.

$$M = r.s \tag{3.1}$$

For instance M can be 667 = 23*29 where r=23 and s=29. But in practice, M is chosen to be a very large value which cannot easily be factorized into r and s. Now i which is the encryption exponent is also chosen such that

$$gcd(i,(r-1)(s-1)) = 1 \tag{3.2}$$

In other for the RSA algorithm to work, equation 3.1 must hold. Using the case of party A, B and C, lets show how RSA can be used to encode a given message. For B to encode a message g intended for A, B would calculate the modulo arithmetic of g^i and M.

$$E = g^i mod M \tag{3.3}$$

where E is the encoded message. On receiving the encoded message E, party A then computes k using 3.4.

$$k = i^{-1} mod (r-1)(s-1). \tag{3.4}$$

The final stage of the decoding is to compute the original message g by the following relation:

$$g = E^k mod M \tag{3.5}$$

Here we present some examples of the RSA with some small numbers. Lets say A chooses an encryption index, i = 17, r=5 and s=11. M becomes M=5*11=55. It is very essential to obtain

$$gcd(i,(r-1)(s-1)) = gcd(17,(5-1)(11-1)) = gcd(17,40) = 1$$ If B wants to encrypt a message g=37, B will calculate

$$E = 37^{17} mod 55 = 27$$

So the encrypted message is 27 is sent to A. Before A decodes the E, A will calculate

$$k = i^{-1} mod (5-1)(11-1) = 17^{-1} mod 40 = 33$$

This can be verified by

$$ik\ mod\ (r-1)(s-1) = 17 * 33\ mod\ 4 * 10 = 561\ mod\ 40 = 1$$

Party A can decode the encoded message by using

$$g = E^k mod M = 27^{33} mod 55 = 37$$

The above described algorithm is already implemented in the openssl utility on most unix based machines including the development workstation used for this thesis. The Openssl was used generate both the public and private keys including the encryption index by invoking the following command on the linux box.

$ sudo openssl x509 -req -days 365 -signkey evans.key -out evans.crt

This Openssl command prouced the files **evans.key** and **evans.crt** which were the private and the public keys respectively. These two files were stored on the server on which the web service

resided. After this stage a configuration file in the web server was edited to support secure connections which is HTTPS. In production environment, it is advisable to consider buying a signed certificate from trusted SSL service providers such as Verisign Both keys were designed to last for 365 days after which it had to be renewed. A transaction was then sent over the HTTPS protocol. The protocol was able to encod the transaction before sending it. The client (a certificate enabled browser, e.g. Mozilla Firefox) is the only entity that knows the private key, hence decoding the sent message. Illustrations of these have been presented in section 4.3.4.2 in chapter 4.

The listing below shows some of the content of the private key which was generated. The content includes the name of the keystore, the date it was created and the type.

Your keystore contains 1 entry evanskotei, Oct 17,

2014, PrivateKeyEntry,

Certificate fingerprint (MD5): EB:28:A5:83:56:58:86:59:A4:CA:7F:3E:0E:A4:C7:22

Chapter 4

Results and Discussion

4.1 Design, Results, Analysis and Discussion

In this chapter a detailed performance analysis is given on the performance of the service. The analysis is presented under the various conditions the web service is subjected to. The section is divided into two sections i.e. the functional and the non-functional tests. The functional test is mostly based on the core functions of the service where as the non-functional test is responsible for the robustness of the service. Notwithstanding these two categories, each category of the test is performed on both the plain and secured HTTP services. This was done in other to identify the advantages of the proposed service. Tables and charts are also included where necessary in other to give a clearer understanding to the results.

4.2 Functional Test

The service was able to pass the functional test. As explained already the functional test deals with the fact that the service would accept the right input data types. It is aform of validation on the web service. The service developed in this thesis accepts only integers casted into string formats. Due to this, if the client passes any string apart from an integer in string format, the web service would respond with a soap fault. In figure 4.1a, it could be seen that the character string **yaw nti**

05120000314 1 tsn6453 was sent directly to the service instead of its integer string equivalence.

The soap fault generated can be seen on the right hand side of figure 4.1a.

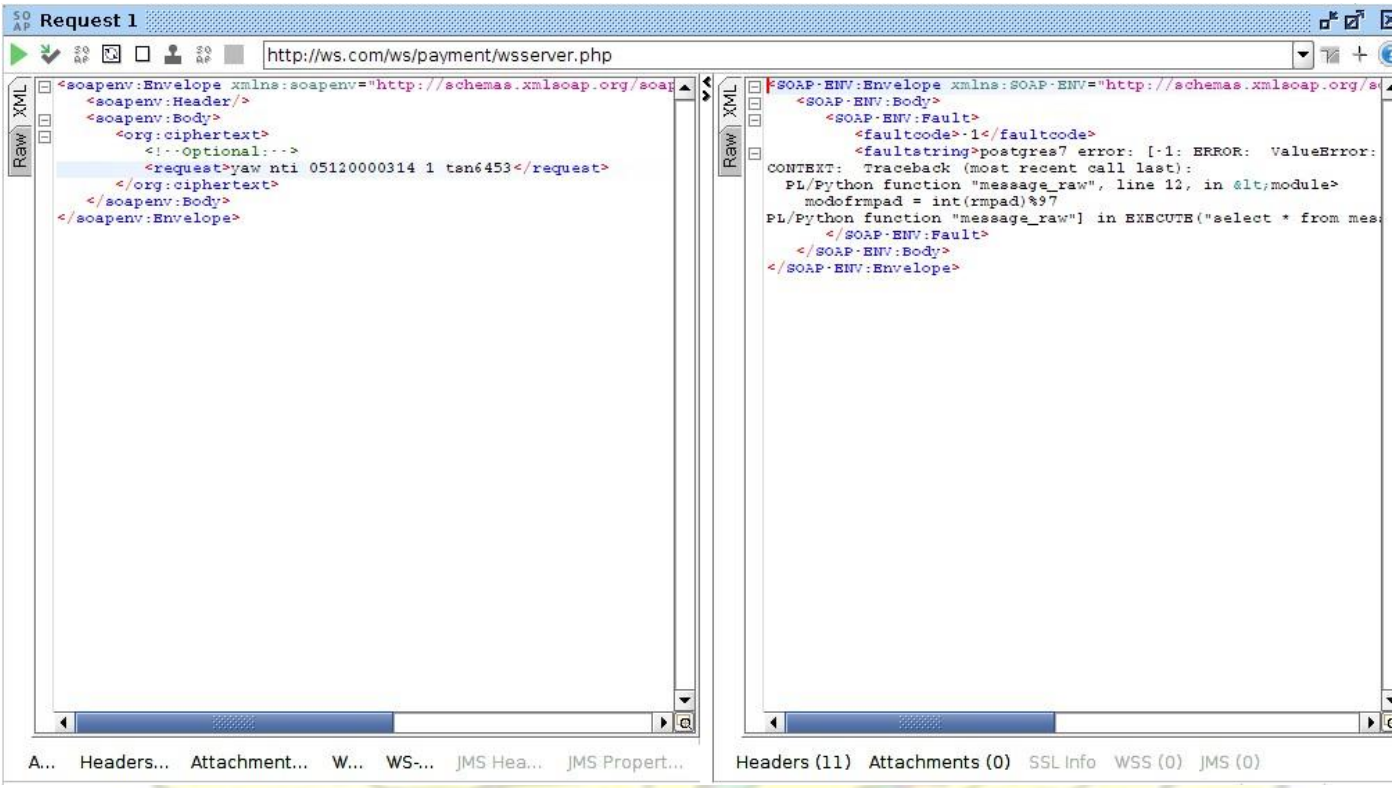

Figure 4.1a: Validation Test a

The service functioned well when an integer string was parsed. In figure 4.1b the integer string **4596727914293915242869270142269126296557994410433306694564 58** of **yaw nti 05120000314 1 tsn6453** was parsed and the web service responded correctly. The response is shown on the right hand side of figure 4.1b.

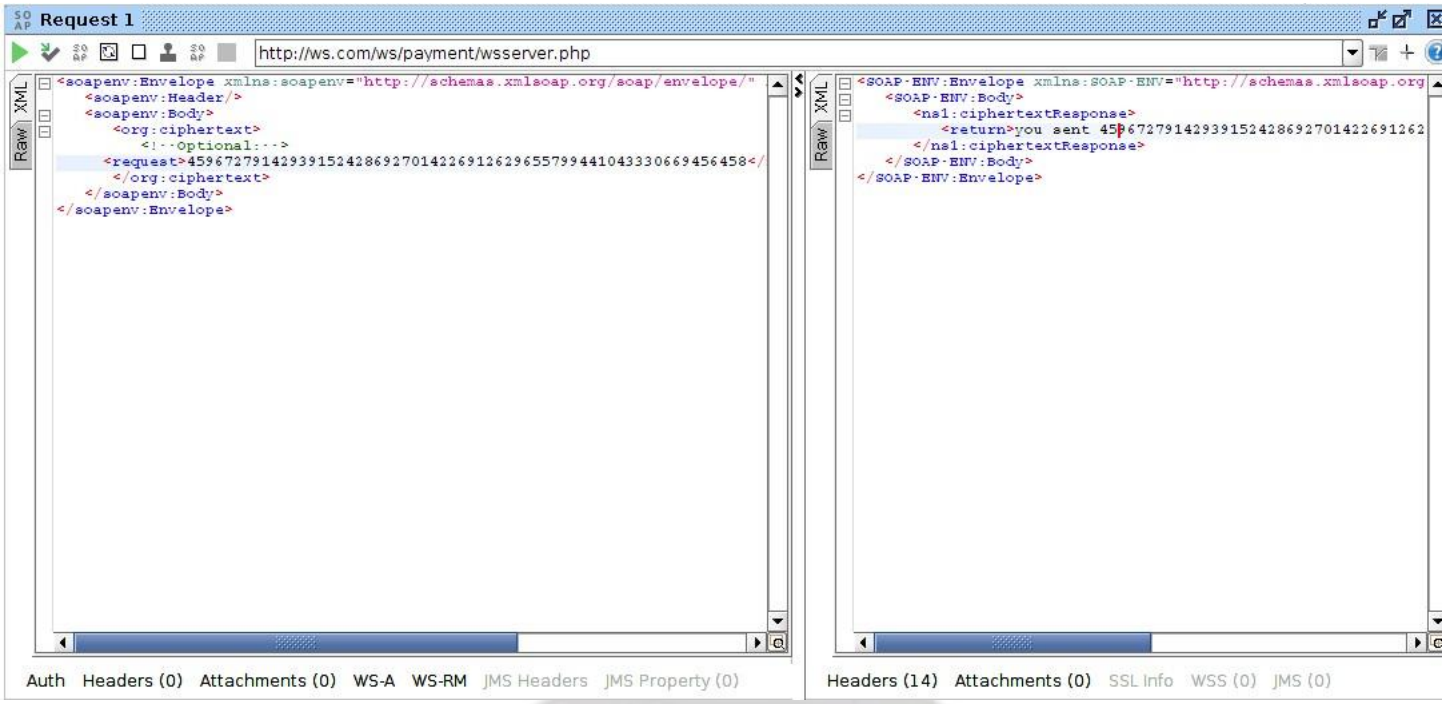

Figure 4.1b: Validation Test b

4.3 Non Functional Test

4.3.1 Load Test

The service which comprises of the secured message sent over HTTP was able to submit an average of 939 records into the database. This implies that approximately 16 records were submitted within every second and since there were five loads, approximately 3 records were sent by each thread per second. So each thread submitted approximately 188 records per minute. The same test was performed for the secured message sent over HTTPS and the raw message sent over HTTP which is synonymous to the existing system. Figure 4.2 shows the mean number of transactions of the three web services invoked for 60 seconds.

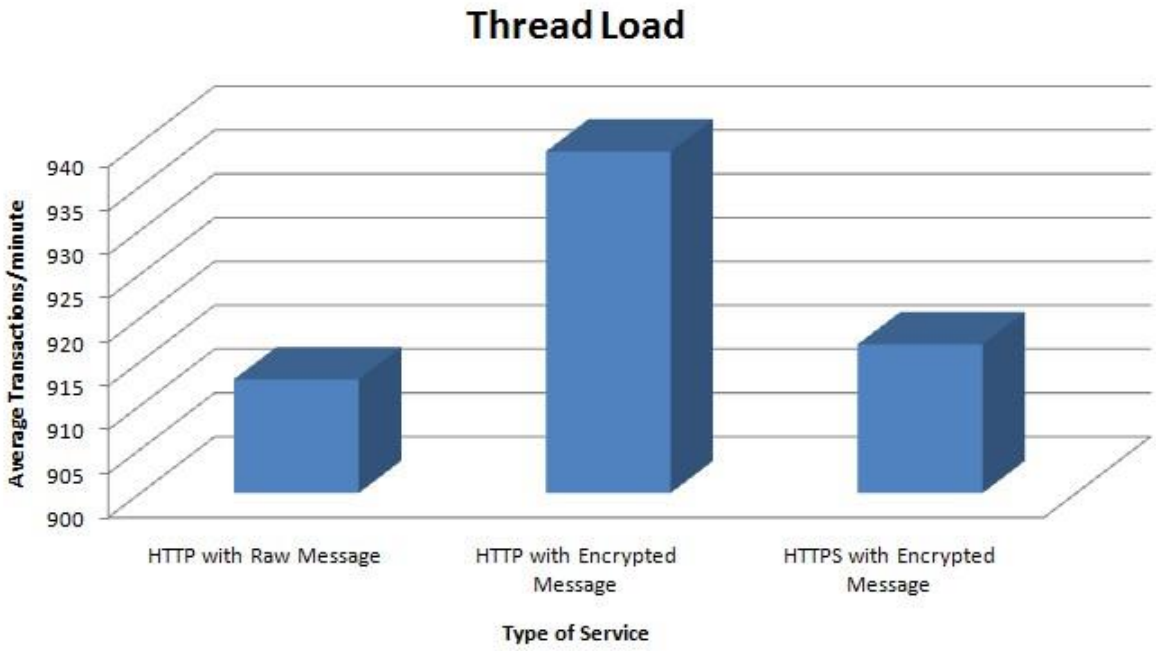

Figure 4.2: Thread Load

Obviously from the table values, we can confidently say that the transactions were not so much affected by the new algorithm which has been implemented. Encrypting the raw message would not affect the number of transactions of the web service. This means that adding the data encoding security feature to the existing implementation will not affect the number of transactions per minute. The existing system measured 913 average transactions, 939 for the HTTP with Encrypted Message and 917 for the HTTPS with Encrypted Message.

4.3.2 Transaction Size

Figure 4.3 represents the transactions in relation to the message size for the HTTP with the encrypted transaction service. The sizes were varied from 1 byte to 40 bytes for mixed character message. For instance a transaction of mixed character character with size 10 bytes was able to register 406 transactions per minute. Similarly a mixed character message with 30 bytes registered

the same number of transactions i.e. 406. From these results it can concretely be said that, the number of transactions do not depend so much on the size of the message being sent.

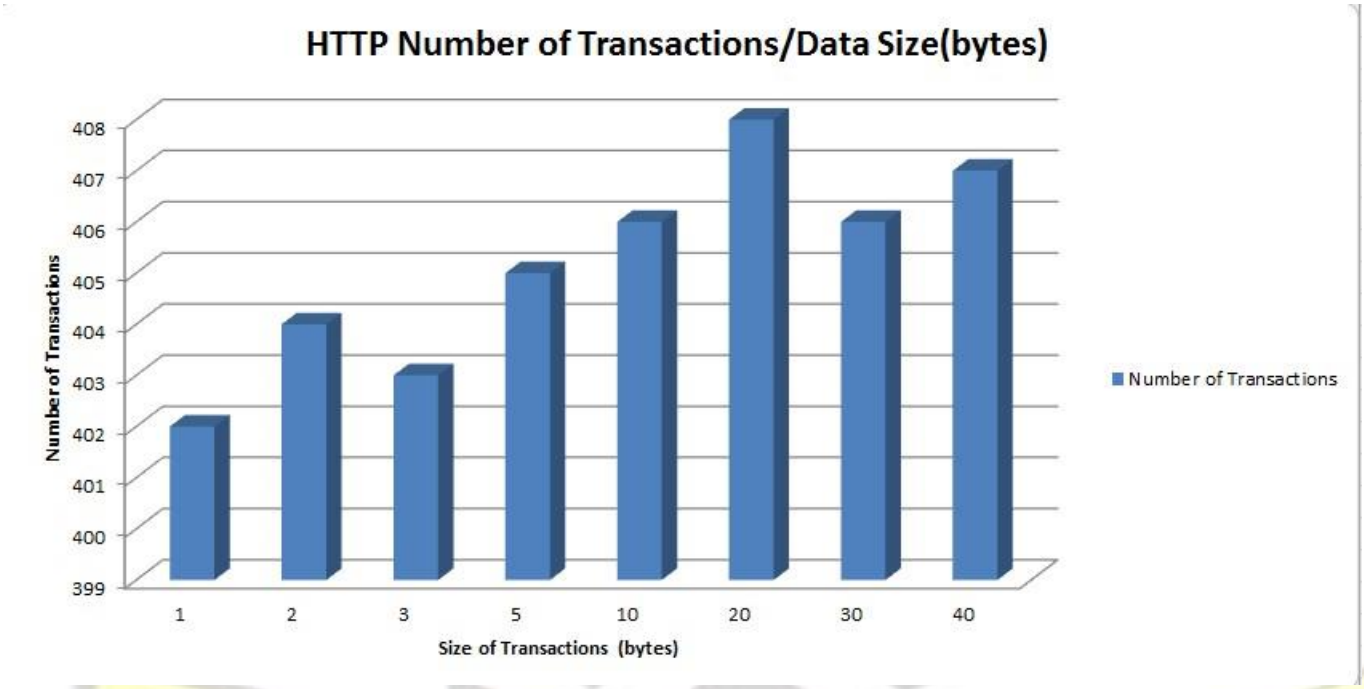

Figure 4.3: Transactions per Minute for Mixed Characters (HTTP)

Figure 4.4 on the other hand represents the transactions in relation to the message size for the HTTPS with the encrypted transaction service.

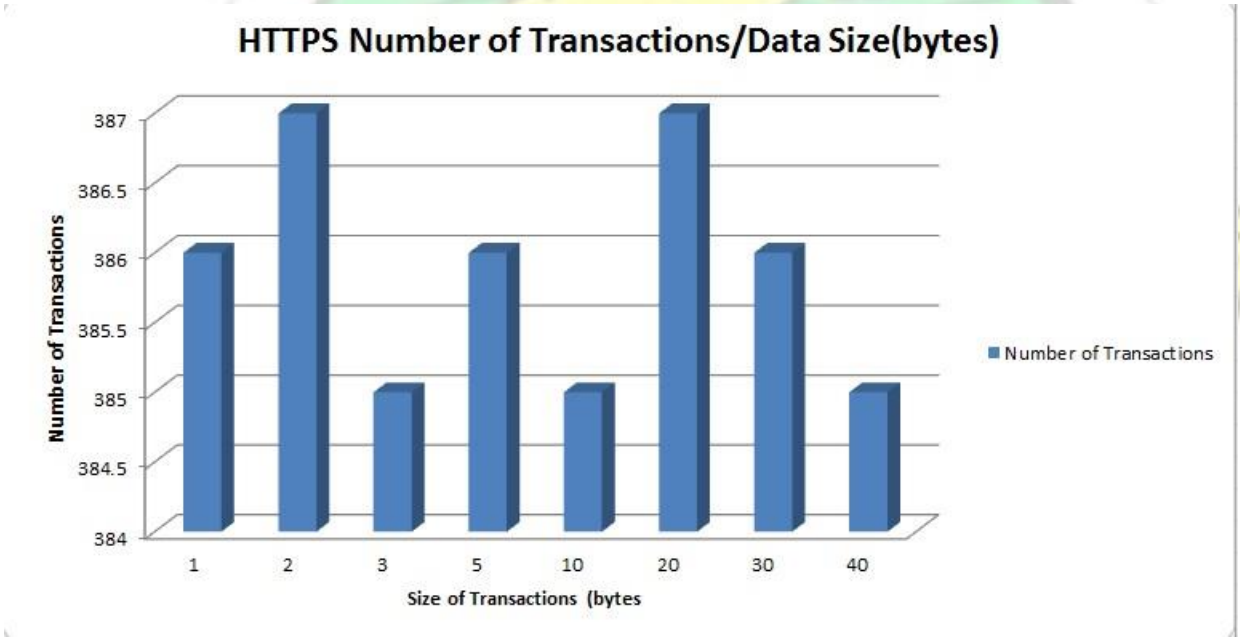

Figure 4.4: Mixed Characters (HTTPS enabled service

For instance a transaction of same character with size 10 bytes was able to register 386 transactions per minute where as that of the mixed characters with 10 bytes registered 385 transactions. Similarly, the number of transactions do not depend so much on the size of the message being sent. Comparing both services, the HTTP enabled web service was able to send more transactions than the HTTPS enabled service. This is attributed to the medium of message transfer. Since the HTTPS transfer protocol has RSA algorithm implemented in it, it used extra time for the computation of the public and private keys as described in chapter 3. Due to this, the HTTPS medium is able to send lesser transactions than the HTTP which has no encryption algorithm.

This can be seen from the fact that a transaction size of 40 bytes registered 387 for HTTPS and 408 for HTTP.

An analysis based on mixed characters are presented here. Since the web service was used to send student data, a transaction with the following entries were sent over all the web services. The transaction consists of a student with name **yaw nti**, student number **05120000314**, paying an amount of **1** Ghana cedi with transaction code **tsn6453**. This transaction was parsed with white spaces. So in all the size of the transaction was 29 bytes. Figure 4.5 shows the results for the 29 bytes message. The results in this section confirms what was discussed in the previous paragraph. The average transaction of mixed characters over HTTPS is 385 where that of HTTP was 404. Since the HTTPS transfer protocol has RSA algorithm implemented, it used extra resources for calculating the RSA based keys.

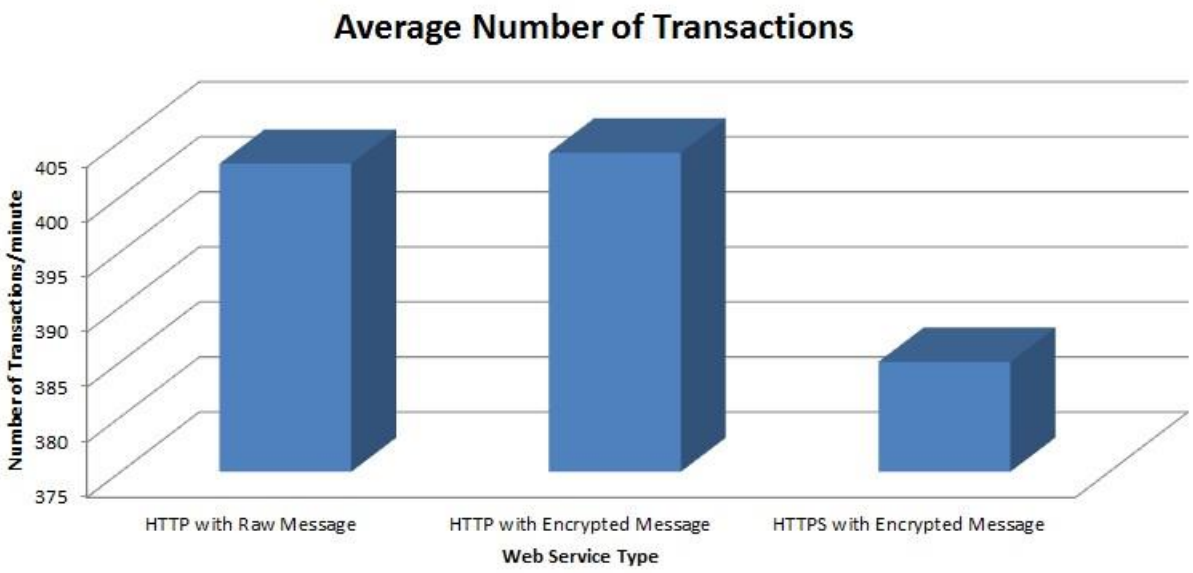

Figure 4.5: Mean Number of Transactions per Minute

4.3.3 Testing Over a Network (LAN)

The same conclusion can be made from the experiment which was performed on the LAN. In sending 29 bytes of message, the average number of transaction registered for the HTTPS based transportation was 353 as compared to that in the HTTP medium which was 401. About 8.3% of the transactions on the HTTPS were dropped when the LAN was introduced.

	HTTP with Encrypted Message	HTTPS with Encrypted Message
Mean	401	353

Table 4.1: Mean Number of Transactions per Minute on LAN

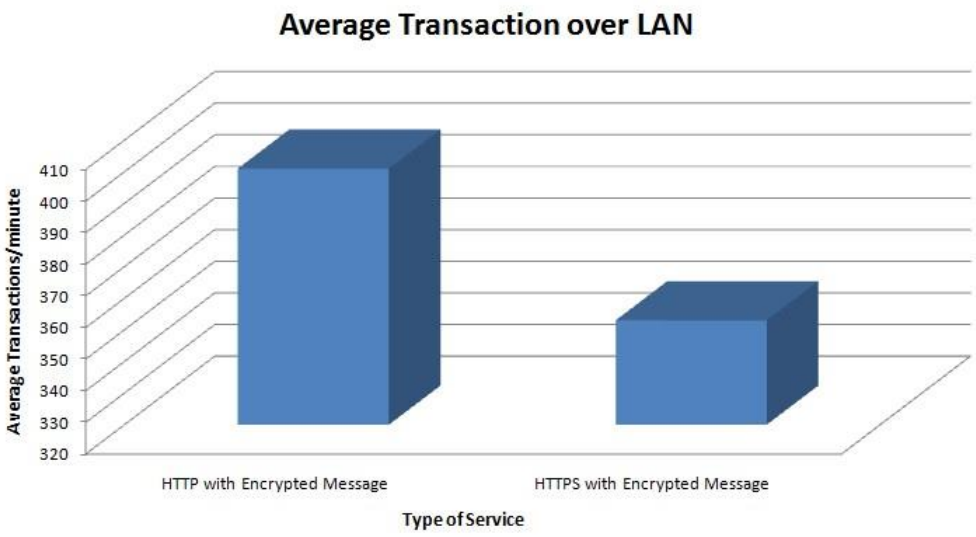

Figure 4.6: LAN Performance

Figure 4.6 also graphically describes table 4.1. More transactions where recorded in the HTTP encrypted mode than in the HTTPS encrypted mode. The reduction in transactions was about 8.3%.

4.3.4 Security Test

4.3.4.1 HTTP Encrypted Transaction Mode

Figure 4.7 shows the logs of the **tcpdump** when the HTTP encrypted message mode was invoked at port 80. In the logs, we can clearly see the files consistuting the web service. In the bottom of the figure 4.7, the requested flagged with **POST /ws/payment/wsserver.php?wsdl** are shown in the logs. A malicious person could siphon some pertinent information in these files and later used them to attack the system.

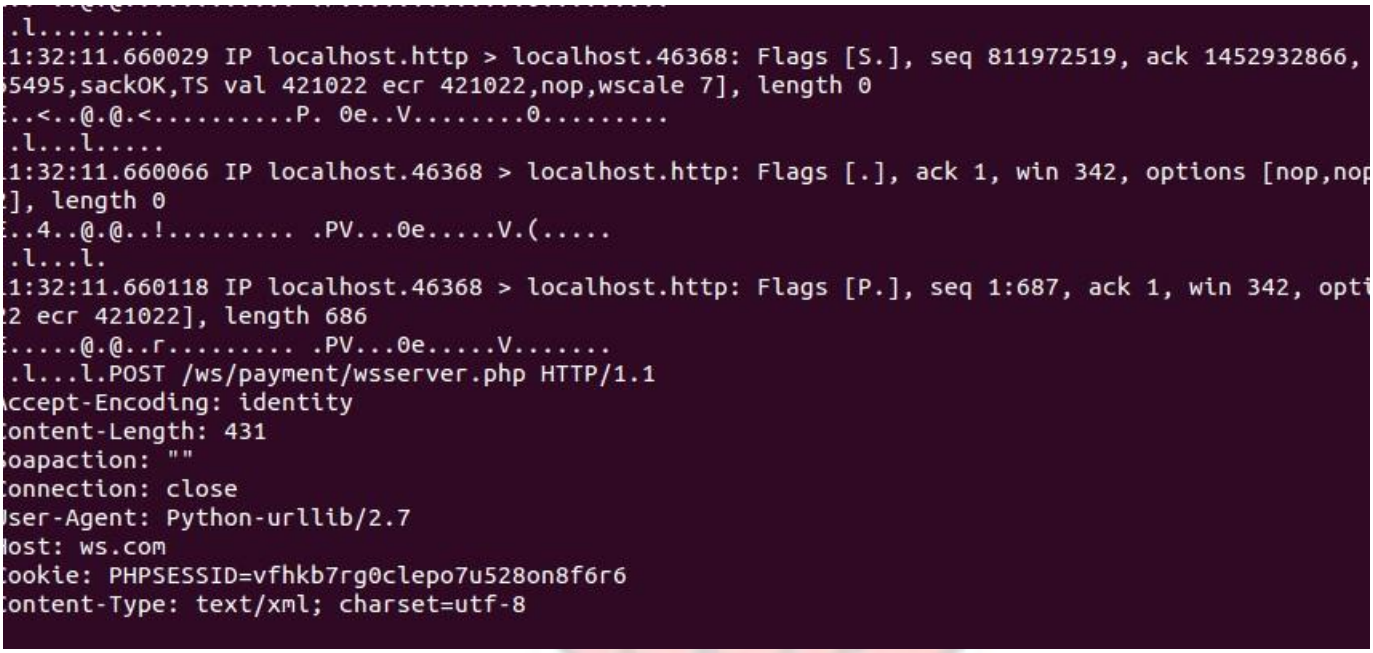

Figure 4.7: Plain Logs showing Requested Files

Figure 4.8 also shows the plain soap request of the service. Clearly, the original information was encoded into a long string of integers as shown at the upper most of the figure. Supposing a an unencrypted message was sent, an attacker could get hold of the messages with ease. But in this case though he could get the transacted message, he would have to decipher the long string **4596727914293915242869270142269126296557994410433306694564 58** to retrieve the original transaction, but how? If even the attacker can decipher the information, it will take a lot of computing time since he does not know how the encoding was done.

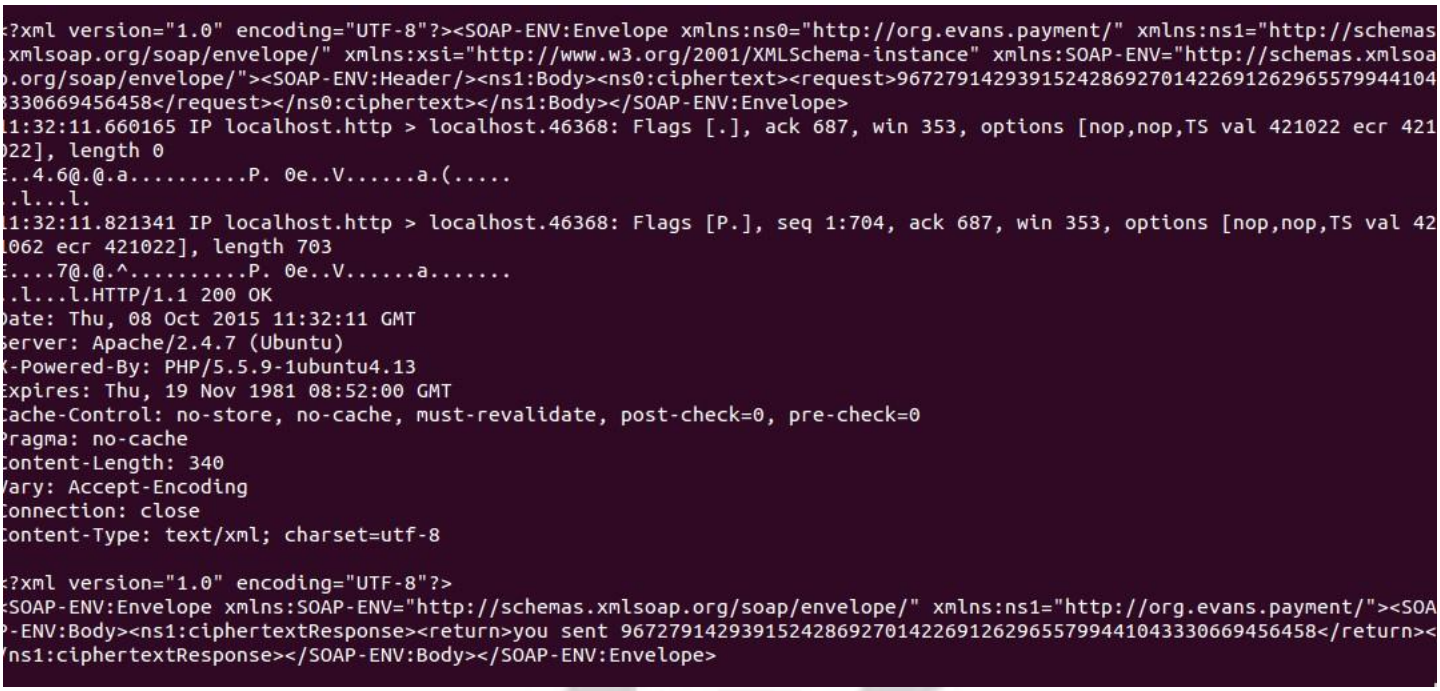

Figure 4.8: Plain Logs of Request

4.3.4.2 HTTPS Encrypted Transaction Mode

Unlike the plain HTTP service, the HTTPS service is invoked at the SSL port number 443, which is the secured port for Apache Web Server. In the generated logs, there are no traces of the soap request and response envelops as the HTTP exhibited. All the messages have been encrypted and only the server and the client can have access to them. This is shown in figure 4.9. In this figure the only messages that can be seen are the acknowledgements from the server on receiving the request from the client. In the figure, the acknowledgements are indicated with dark backgroud.

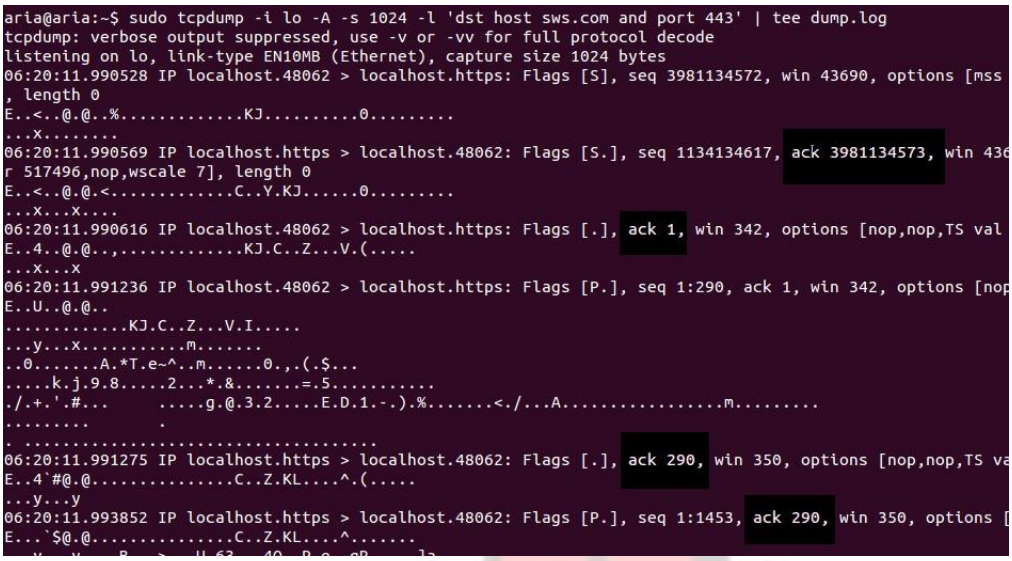

Figure 4.9: Logs of Secured Service

Figure 4.10 is a portion of figure 4.9. This actually depicts the encrypted data being transmitted through an encrypted channel. The alphabets and periods in the figure are all essential information that have been encoded.

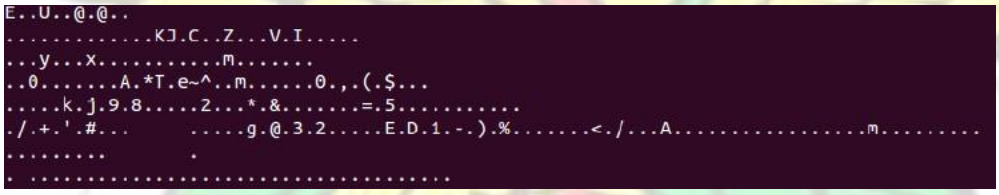

Figure 4.10: Logs of Secured Service

The certificate issued during this transaction contains the entries shown in the listing below. It shows the public key that was used to encrypt the transaction. It also shows the signature algorithm used in the encryption. In the listing the modulus,

CipherSuite : *TLS _ECDHE RSA WITH AES _*128 *CBC SHA*

PeerPrincipal : *CN = evanskoteithesis,OU = ICT,O = ICT,L = Ksi,ST = Ksi*

*PeerCertificate*1 :

Subject : *CN = evanskoteithesis,OU = ICT,O = ICT,L = Ksi,ST = Ksi,C = GH*

SignatureAlgorithm : *SHA*1*withRSA,OID* = 1.2.840.113549.1.1.5 *Key* : *SunRSApublickey,*1024*bits*

modulus : 13625171295312304454398306085473660972445773624597100702687788389116090565859202344211905985379325763318490097043607622692781209211487348771984078763

480203312661777424334731676318553893450106550703286184966007874454270974453543 13

909620596877215832037051238477796969596820890697024950158778247357986634613350 23

which is a very large integer is used as the public key. It would take a huge computing resources to be able to easily factorize this number. Also shown here beloww in this listing is the private key signature.

publicexponent : 65537 *Signature*

:

0000 : 845*EEE*42499630*F*2168*EDF*144*B*1*DB*754.·*BI*.0.....*K*..*T Y*

*ourkeystorecontains*1*entry*

*evanskotei,Oct*17,2014,*PrivateKeyEntry,*

Certificatefingerprint(*MD*5) : *EB* : 28 : *A*5 : 83 : 56 : 58 : 86 : 59 : *A*4 : *CA* : 7*F* : 3*E* :

0*E* : *A*4 : *C*7 : 22

4.3.4.3 HTTP Raw Transaction Mode(Existing System)

Figure 4.11a,4.11b and 4.11c all shows the logs from the existing system when a transaction is sent through it. In figure 4.11a, the transactional request, **yaw nti 05120000314 1 tsn6453** is shown plainly in the logs.

```
<?xml version="1.0" encoding="UTF-8"?><SOAP-ENV:Envelope xmlns:ns0="http://org.evans.payment/" xmlns:ns1="http://schemas
.xmlsoap.org/soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENV="http://schemas.xmlsoa
p.org/soap/envelope/"><SOAP-ENV:Header/><ns1:Body><ns0:ciphertext><request>yaw nti 05120000314 1 tsn6453</request></ns0:
ciphertext></ns1:Body></SOAP-ENV:Envelope>
13:13:45.184839 IP localhost.http > localhost.46453: Flags [.], ack 658, win 352, options [nop,nop,TS val 1944403 ecr 19
44403], length 0
E..4.)@.@.............P.u...z.......`.(.....
...S...S
13:13:45.355533 IP localhost.http > localhost.46453: Flags [P.], seq 1:675, ack 658, win 352, options [nop,nop,TS val 19
44446 ecr 1944403], length 674
E....*@.@.............P.u...z.......`.......
```

Figure 4.11a: Client Soap Request

Figure 4.11b depicts the HTTP response to the request. the data sent was which is exposed to any attacker listening to that particular port could retrieve the information. It indicates the web server which is Apache version 2.4.7 running on Ubuntu Operating System. It also shows the content type of the request which is xml being the soap request.

```
...~...SHTTP/1.1 200 OK
Date: Thu, 08 Oct 2015 13:13:45 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Powered-By: PHP/5.5.9-1ubuntu4.13
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Content-Length: 311
Vary: Accept-Encoding
Connection: close
Content-Type: text/xml; charset=utf-8
```

Figure 4.11b: Response from HTTP

In the figure 4.11c, the soap response from the server is shown. The server response with with the same message which was sent. This transactions are exposed to any attacker who is listening to the port on which the web services is hosted. He could retrieve the transactions and later alter it.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="http://org.evans.payment/"><SOA
P-ENV:Body><ns1:ciphertextResponse><return>you sent yaw nti 05120000314 1 tsn6453</return></ns1:ciphertextResponse></SOA
P-ENV:Body></SOAP-ENV:Envelope>
```

Figure 4.11c: Server Soap Response

4.4 Summary

In this chapter, various investigations have been made on an improved way of data communication through web services. A new system had been developed from an existing one. Both systems were subjected to various test scenarios to see how best the new system would behave. The analysis showed that the existing system (HTTP with Raw Message) had an advantage over the new one in respect of the number of transactions per every 60 seconds. Though this was its advantage, the reduction was only about 8.3% of transactions. Security-wise, the new system demonstrated tremendously well since it was able to encode essential data sent through it. Hence the new system sacrificed about 8.3% of its transactions to compensate for securing the data.

Chapter 5

Conclusion

5.1 Conclusion

In this thesis, we showcased a secured way of transferring highly sensitive data through Hyper Text Transfer Protocol (HTTP) by implementing two kinds of security levels. One being an algorithm for data encryption and securing the transmision mode.

The results proved that encrypting data over the internet and also encrypting the transfer protocol has no significant effect of the data that is transfered..

5.2 Summary of Findings

This thesis was to examine the performance of web service instances and its deployment between two actors i.e. Kumasi Polytechnic and its financial agents. Typically this study was to come up with an effective and efficient mode of school fees transaction between the school and its banking partners. In the earlier chapters of this thesis a background study was made about XML, SOAP and the concept of web services. Carrying out this thesis has unleashed an in-depth understanding on how to develop and deploy web service instances to perform various activities.

5.2.1 Support for Heavy Load

The payment web service actually supported much load as expected. From the analysis it was noticed that the payment service was able to carry a heavy load of users at the same time. This makes the service robust enough to support real world transactions without breaking down.

5.2.2 Response Time

In addition, the response time of the payment service was very convincing since it took few seconds to transmit the transactions. To elaborate much on this, this study had in mind that the existing mode of transaction took so long i.e. about a day or two before the institution gets access to student fees data from its banking agents. As it has been stated already, this hinders students registration and other activities on Kumasi Polytechnic Campus. The payment web service has proven so efficient that the time between transactions is cut down drastically to a few seconds. What this means is that, students can actually register a few minutes after paying their school fees without queuing unnecessarily.

5.2.3 Managing Internet Resource

In contrast to other mode of transactions such using an API from the banking agent to monitor fees payment, the payment web service help both parties to manage some network resources well. Imagine a situation where by the banking agent providing an API for the school. This means that, the school has to always log onto the banks systems in other to access school fees. This can make the Internet up-link of the banking agent ran out allowing very slow connections from other users. With this payment notification web service, no or very infinitesimal up-link is involved. Here both

parties do not fully log onto any system which necessitates high up-link but only methods are exposed to the Internet.

5.2.4 Secured

Lastly this mode is also secured since the entire systems are not exposed to network. Here we exposed a small portion of a software which is just a method or function in the software. In addition to the above, the url for the transaction is secured and the messages are encryted to prevent man-in-the-middle-attack.

5.3 Recommendation and Future Work

All the experiment performed in this thesis represents latency response time and load support of the web service developed. These tests mentioned above help us to strengthen the cognition towards service performance and their disadvantages. Hence, we can seek after possible solutions to enhance their performance. In a word, there are still many things to do towards the research on Web Services. We recommend that a web service should be deployed on a windows server and tested.

Bibliography

1. Andurkar, A. D. 2012 Column-Oriented Database Implementation in PostgreSql for ImprovingPerformance of Read-Only Queries, Masters Thesis.

2. Chen et al 2006 An efficient approach to web services discovery and composition when largescale services are available. In Proc. of IEEE Asia-Pacific Conference on Services Computing (APSCC 06), pages 3441, 2006.

3. Chinthaka, E. 2006 AXIOM- Fast and Lightweight Object Model for XML.

4. Davis, T. 2003 RSA Encryption October 10, 2003

5. Du, G. 2004 Web Services Performance Study, MSc Thesis

6. Felipe, L. O. 2010 Design and development of a REST-based Web service platform for applications integration February 2010 Thesis

7. Gregor et al 2003 Enterprise Integration Patterns: Designing, Building, and Deploying MessagingSolutions. and Addison-Wesley, 2003.

8. Gross, E. 2013 *CICS Web Services as a Provider and Requester*, Circle Software Incorporated

9. Guruduth et al 1999 A Case for Message-Oriented Middleware. In Proceedings of the 13th International Symposium on Distributed Computing, 1999.

10. Haas, A. Brown, A. 2004 *Web Services Glossary* http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice.

11. Huhns et al 2005 Service-Oriented Computing: Key Concepts and Principles. IEEE InternetComputing, 9(1), 2005.

12. Jayasinghe et al 2011 Apache Axis2 Web Services, 2nd Edition. ISBN 978-1-849511-56-8 PacktPublishing House 2011

13. Jones, S. 2005 Toward an acceptable definition of service [service- oriented architecture]. IEEETransactions on Software, 22(3):8793, May-Jun 2005.

14. Kyrnin, J. 2014 Who Uses XML? http://webdesign.about.com/od/xml/a/aa060401a.htm Accessed: June 22, 2014

15. Leitner, P. 2007 The Daios Framework - Dynamic, Asynchronous and Message-oriented Invocation of Web Services 18th September 2007 Masters Thesis

16. Manoj et al 2003 A Literature Review on Trust Management in Web Services Access ControlInternational Journal on Web Service Computing (IJWSC), Vol.4, No.3, September 2013

17. Pelz-Sharpe, A. 2010 What is XML and Why ShouldCompanies Use It?

http://www.inc.com/guides/2010/04/why-companies-should-use-xml.html Accessed: June 21, 2014

18. Ramesh et al 2003 Developing Java Web Service Architecting and Developing Web ServicesUsing Java 2003

19. Russel, R.2014 Which style of WSDL should I use? http://www.ibm.com/ developerworks/webservices/library/ws-whichwsdl/. Visited: 2014-09-19.

20. Simmonds, J. 2011 Dynamic Analysis of Web Services, PHD Thesis, University of Toronto

21. Singh, R. R. 2004 *Collaborative Urban Information Systems: A Web Services approach*, Doctors Thesis at the Massachusetts Institute of Technology.

22. Tapang, C. C. 2001 Web Services Description Language (WSDL) Explained, Infotects, July2001.

23. The Open Group, 2014 The Open Group. The SOA Work Group.

http://www.opengroup.org/subjectareas/soa Accessed: May, 2014.

24. The Tutorials Point 2014 The Tutorials Point http://www.tutorialspoint.com/uddi/uddi overview.htm

Accessed: May, 2014

25. OASIS, 2004 Organization for the Advancement of Structured Information Standards, OASIS,2004 Introduction to UDDI:I mportant Features and Functional Concepts. October, 2014.

26. postgresql.org 2014 postgresql.org http://www.postgresql.org/ Accessed: August 10, 2014

27. Web Services Security 2008 Government of Hong Kong Web Services Security.

28. webservices 2014 webservices http://www.w3schools.com/webservices/ws wsdl _ document.asp

Accessed: September 01, 2014

29. O3b Networks 2011 What is Network Latency and Why Does It Matter?

30. Oracle Fusion 2012 Oracle Fusion Middleware Security and Administrator's Guide for Web Services.

31. World Wide Web Consortium 2002 WSDL, Web Service Description Language.

http://www.w3.org/TR/wsdl, 2002. Visited: 2007-07-31.

32. World Wide Web Consortium (W3C) 2006. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer - W3C Candidate Recommendation 27 March 2006.

http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/, 2006. Visited: 2014-09-19.

33. Web service interoperability organization (WS-I) 2014 Basic Profile Version 1.2.

http://www.ws-i.org/Profiles/BasicProfile-1.2.html. Visited: 2014-09-19

Appendix

Ciphering Codes

```
import string import random

from fractions import gcd

import time


print "Welcome to Messsage Cipher!"


#key = input("Please enter the encryption key: ")


raw_message = raw_input("Input the payment information in this order(name,studentno,amonut,transid) : ")

key=45 message = string.lower(raw_message)    ### inpute message to lower case code = "" chars =

'abcdefghijklmnopqrstuv wxyz4728139506' for letter in message:          ### loop through original message

index = string.find(chars, letter) ### get index of letters from chars number = str((index + 1 + key)%37) ###

since its a 37 alphabet chars, add one, key to index and compute mod 37 to string code += "%s " % number

### form string of new indces with spaces


sd = code.split(' ')        ### convert them to array strings with comma separation sd.pop()        ###

remove the last space sd = map(int, sd)  ### convert the array strings to integer array

msgint=0

modw=95

for i, item in enumerate(sd):      ### loop through the integer array where i is the index of each item

  msgint += item * pow(modw,i)
```

Deciphering Codes

```
modw=95
    retd = "" codio = "" rmpad = raw_message[:-
    2] pad = int(raw_message[len(rmpad):]) if
    modofrmpad == pad: while int(rmpad) >=
    modw: ret = str(int(rmpad)%modw) rmpad =
    int(rmpad)/modw retd += "%s " % ret
     reta=(retd + str(rmpad)) reta =
     reta.split(' ') sd = map(int,
     reta) codio=""
     for i in sd:
      pos = (i + 36 - keyy)%37 codio +=
      "%s" % chars[pos]
```