UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

A Model That Facilitates Secure Transmission of Examinations Data

Eunice Njoki Mwai

Reg. No: P58/62217/2010

A research project submitted in partial fulfillment of the Requirements for the award of Degree of

Masters of Science in Computer Science at the

School of Computing and Informatics

Of the University of Nairobi

June 2014

DECLARATION

This project is my original work and has not been presented in any University or college in any country.

Signed by é é é é é é é é é é é é é é é é é Dateé é é é é é é é é é é é

Eunice Njoki Mwai

P58/62217/2010

APPROVAL

This research project has been presented for examination with my approval as University Supervisor.

Signed by é é é é é é é é é é é é é é é Dateé é é é é é é é é é é é

Mr. Lawrence Muchemi

Lecturer

University of Nairobi

School of Computing and Informatics

DEDICATION

This project is dedicated to my family.

ACKNOWLEDGEMENT

Let me take this opportunity to express my sincere gratitude and to thank my sponsor and all those who contributed directly to ensure the success of this project.

First and foremost, I would like to convey my special thanks to Mr. Lawrence Muchemi, my supervisor for the advice and guidance he gave me during the development of this project. I also thank the panel members: Dr. Andrew Mwaura, Dr. Opiyo and Ms. Christine Ronge for sitting in the panel that made this project a success.

Iôm indebted to members of my family, Father Mr. Joseph Mwai, The late Mother Mrs. Mary Wambui and all the other family members that gave me support while undertaking this project.

My other dedication goes to all mastersô 2012 class and others who have sacrificed in one way or another that have made it possible for me to finalize this project.

Last but not least, I would like to return praise to our almighty God for the free life He has given me, wonderful opportunity that I have had and the knowledge I shared with others.

ABSTRACT

The tremendous growth of the digital data environments requires new architectures for security services prompting a need for secure transfer of data especially across institutions of higher learning. However, there are fundamental security concerns to be considered before data can be transferred and these include attacks both passive (unauthorized access) and active attacks (destruction), editing and data reproduction. This research brings forth a model that ensures integrity of data by minimizing unauthorized access attacks. This research uses symmetric key encryption algorithm, in which same structure of encryption and decryption is used. The model provides an extension called a guard where the algorithm uses key generation method by random number for increasing efficiency of algorithm. The internal key generator will store the sender key and send to the receiver end by other path for preventing harmful attacks on the model.

To evaluate the developed security model, a prototype was developed using java. The prototype had inbuilt security logs to access who or what was being accessed in the prototype. A statistical method was used to determine the sample size of the security logs collected by the prototype for a period of one month. The logs were analyzed before the key generator was implemented and after implementing it. The analysis showed the Number of login failures after the key generator was implemented was 105, operation problems were 20 and potentially dangerous files were 17. An analysis on the number of unauthorized access before and after implementing the prototype was done and this shows that after the key generator was implemented, the number of users trying to access the prototype was reduced. Also the number of users who managed to access the prototype and tried to decrypt the uploaded text files had also reduced. These experiments showed that AES based model with key generator has improved the security aspect of the prototype and therefore text files can be uploaded in an encrypted format to ensure that unauthorized users donôt get access to the text files. This analysis shows that it is possible to confidently upload encrypted documents.

This model provides a basic design allowing encryption and decryption to be performed by combining a high quality scrambling technique with a strong encryption/decryption mechanism. This research has brought forth a secure model for transmission of files with an emphasis on examinations related files.

TABLE OF CONTENTS

List of Tables

List of Figures

Abbreviations and Acronyms

JKUAT	Jomo Kenyatta University of Agriculture and Technology
NIST	National Institute of Standards and Technology
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
S-HTTP	Secure Hypertext Transfer Protocol
FTP	File transfer protocol
DES	Data encryption standard
AES	Advanced encryption standard
IDEA	International data encryption algorithm
PGP	Pretty good privacy
FIG	Figures
IBM	International Business Machines
RSA	Ron Rivest, Adi Shamir and Leonard Adleman
3DES	Triple Data encryption standard
KEK	Key Encryption Key
TEK	Transfer Encryption Key

CHAPTER ONE
INTRODUCTION

1.1 Background

The importance of information and communications systems for society and the global economy is intensifying with the ever-increasing value and quantity of data being transmitted and stored on those systems. However, channels of data transmission face increasing vulnerability to a variety of threats, such as unauthorized access and use, misappropriation, alteration, and destruction. Proliferation of computers increased computing power, interconnectivity, decentralization, growth of networks and the number of users, as well as the convergence of information and communications technologies, while enhancing the utility of these systems, also serve increase system vulnerability. Security of information and communications systems involves the protection of the availability, confidentiality and integrity of both the systems and the data that is being transmitted or stored along such systems (Sayed Tathir Abbas, 2013).

Cryptography makes web sites and electronic data safe for storage as well as transmissions over available networks. For a web site to be secure, all of the data transmitted between the computers where the data is kept and where it is received must be encrypted. This allows people to do online banking, online trading, and make online purchases with their credit cards, without worrying that any of their account information is being compromised. As more businesses get conducted over the Internet, the need for protection against fraud, theft, and corruption of vital information increases. Cryptography is therefore paramount to the continued growth of the Internet and electronic commerce. (Krishna Kumar Pandey, 2013)

Public-key cryptography schemes are widely adopted wherever there is a need to secure or authenticate confidential data on a public communication network. When deployed with sufficiently long keys, these algorithms are virtually unbreakable. Today, advances in semiconductor technology and hardware design have made it possible to execute these algorithms in reasonable time even on consumer systems, thus enabling the mass-market use of strong encryption to ensure privacy and authenticity of individualsô personal communications. Consequently, this transition has enabled the proliferation of a variety of secure services, such as online banking and shopping. Examples of consumer electronics devices that routinely rely on high-performance public key cryptography are Blu-ray players. (Pellegrini, 2006)

Since entering into 21st century, there has been a rapid increase of computer network development; information technology is now more and more blended into our daily life at the coming of electronic era. Google and IBM jointly proposed the concept of data sharing in 2007 (Wang Bo al, 2011). The sharing of resources reduces the cost to individuals. Today due to the advancement of technologies and high-speed Internet facilities, it is possible to realize data sharing services. Data sharing is an Internet based model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) (Commerce, 2007) This technology allows for much more efficient computing by centralizing storage, memory, processing and bandwidth. The objective of this research is therefore to ensure security in the transmission of examination data over insecure Internet channels.

In Internet services, confidentiality, integrity and availability are the key challenges. The way to access any services over Internet is through web browser. Web browsers typically use HTTPõS protocols such as HTTP, HTTPS and S-HTTP. HTTP helps to communicate with web servers. In general, in HTTP, sending and receiving information between web server and web browser happens without encrypting messages (Yin, Issues of security protocols in relation to encryption, 2003). However for sensitive transactions, such as Internet e-commerce or online access to financial accounts, the browser and server encrypt this information, referred as HTTPS (He, 2007). HTTPS has been designed to withstand data hacking and provides data confidentiality ((He, 2007). HTTPS also faces some challenges such as: Complex encryption method, Browser incompatibility in decrypting messages, User needs to wait for long to get session ends (Xubin He, 2007), Man-in-the-middle attack (Moxie Marlinspike, 2009), Eavesdropper attack (Xiaoli Yu, 2003).

For transactions carried out in todayõs electronic world, the scenario with the adversary is, unfortunately, the right scenario. An electronic message passes through a number of routers on its way from sender to recipient, and some of these routers may be controlled by malicious parties, and may run malicious code. This may alter the message while it is in transit. On the other hand, such routers may leave the data intact, but nothing prevents other users from reading it. Hence the need to secure the integrity and privacy of data communication However, solving the issue of network security alone does not answer other questions such as: Do the users trust each other? Is there a chance that one of them is behaving maliciously,

and what damage can it cause the other party. (Lysyanskya, 2002).

This research is organized as follows:

Section II explores the current examination system and data sharing background, Internet and its deployment types.

Section III highlights the identified security issues faced today, the encryption algorithm implemented currently and the proposed model for the improved encryption algorithm.

Section IV Test the prototype and evaluate the prototype.

Section V concludes this research along with the future recommendations.

1.2 Statement of the Problem

There is need for a secure model that facilities secure transmission of examination data from one person to another. In transmission of examination data over the Internet, the data is transferred between the server and client. Data is secured in the transmission channel using a strong encryption algorithm and server where it is stored, based on usersô choice of security method. This model prevents unauthorized entries to server, which stores examination data within the examination department in the Universities.

With the increase of geographically distributed campuses within Universities, there is need to secure the data that is transferred from one campus to another. However there are limited measures when it comes to securing this data. Examination data is very sensitive and access to it by unauthorized personnel might result to stakeholders losing confidence in the examination process. This has necessitated the need for a secure way to transmit this data from one computer to another computer.

This research developed a secure model that addresses areas of examination data encryption that are important for education sectors especially universities. The prototype developed from the model encrypts and uploads text files and confirms the vulnerability of the encrypted files. The secure Examinations management prototype provides a secure channel, where text files were uploaded on the prototype using dynamic symmetric keys. This means that the key generator in the AES could be relied upon as an encryption algorithm to upload examination documents in a safe way. This model will prevent unauthorized access to systems, which stores examination data within the examination department in the Universities hence improve the confidence in this department.

1.3 Objectives of the study

The main objectives of this research are as follows:

1. Analysis of the current Examination System Management.
2. Designing a secure model for transmission of data within the examination system management.
3. Development of a prototype based on above model.
4. Evaluation of the performance of the model based on prototype experiment.

1.4 Research Questions

This research project seeks to ask the following research questions based on the objective of the research objectives. Below are the research questions.

1. How does the Examination systems currently use, store and protect documents to prevent a data breach?
2. What are the factors to consider when implementing security in such an Examination system?
3. What are the current existing encryption algorithms?

1.5 Significance of the Study

The findings of this research project are of great importance to the Universities as the model developed will prevent unauthorized access to systems, which stores examination data within the examination department in the Universities hence improve the confidence in this department.

The study findings are expected to contribute to the knowledge on implementation of a secure Examination system where Universities can be sure that examinations are handled in a secure manner putting in mind standards such as confidentiality, Integrity and availability. This research model comes with a key generator that improves the security on the existing AES encryption algorithm and hence makes the model more secure than the current existing encryption models.

1.6 Scope and Limitations of the Study

This research covers the development of a prototype for secure transmission of the examination documents. The prototype encrypts and uploads text files and then confirms

vulnerability of the encrypted files within the JKUAT University and all its distributed satellite campuses.

1.7 Justification of the Study

The research comes up with a prototype based on a model which encrypts and uploads text files and then confirms the vulnerability of the encrypted files. This model aims at finding a solution that improves the security of the examination data been transmitted over insecure Internet. This captures the examination data flow from the time the examination is uploaded to the examination portal to the time the examination officer is able to access it. This therefore leads to improved efficiency and security of Examinations. The University benefits greatly in improving the integrity of the examination process and hence improve the overall image of the University.

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

Today security is the main concern about the transmission of data. We need a cryptosystem which ensures that our data will be secure during the transmission over the network. In secured communication the information is converted from intelligible to unintelligible structure using certain coding operation at the transmitter. There are some techniques that are used for making the data secure during conveying information over the network one these are known as encryption and decryption. (Brindha, 2014). This research project brings forth a model that ensures integrity of data by minimizing unauthorized access attacks especially the ones over the Internet. This research uses enhanced AES symmetric key encryption algorithm based on block cipher generating mechanism in which same structure of encryption and decryption is used. In the review, the main concepts of security standards and terms are discussed in depth. Cryptography is reviewed in depth and its importance in relation to data security, Transmission security is also discussed and the different types involved. Encryption is also discussed in detail and here, the two types of encryption i.e. symmetric and asymmetric are explained in detail and the mechanisms involved in performing them. Classes of security attacks are also mentioned. The research reviews the existing encryption methods and explains each and every one of them. AES encryption algorithm is analyzed in detail because the research improves on it by adding a guard which generators keys dynamically. After that the current encryption method is also discussed and the direction in which the research is heading. Finally the Model for secure transmission is discussed.

2.2 Cryptography

The emergence of the Internet as a trusted medium for commerce and communication has made cryptography an essential component of modern information systems. Cryptography provides the mechanisms necessary to implement accountability, accuracy and confidentiality in communication. (Jerome Burke, 2009). The quality of a cipher is judged by its ability to prevent an unrelated party from determining the original content of an encrypted message. (Jerome Burke, 2009). Cryptography is the method that allows information to be sent in a secure form in such a way that the only receiver is able to retrieve this information. (krishna kumar, 2013). A cryptographic algorithm or cipher is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a

key ï a word, number or phrase ï to encrypt the plaintext. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things; the strength of the cryptographic algorithm, plus all possible keys and all the protocols that make it work comprise a cryptosystem. (Ayushi, 2010)

2.3 Transmission Security

Transmission security is the capacity to send a message electronically from one computer system to another while ensuring that only the intended recipient receives and reads the message and the message received is identical to the message sent. The message would not be identical if it was altered in anyway, whether transmitted over faulty channels or intercepted by an eavesdropper. Information is considered intercepted when someone other than the intended recipient receives the information e.g. by electronic eavesdropping or by using the recipient's password. (Denning, 1982). This can occur anywhere, including a chat room or through an e-mail exchange.

2.3.1 File Transmission

File transfer involves one computer transferring a block of data and expecting nothing in return other than acknowledgment of reception. With file transmission, only the file to be transferred is encrypted. Anyone who intercepts the transfer would only know that something had been transferred but cannot interpret it. Encryption takes place at any time before transfer. Types of file transfer include transmission of files through FTP (file transfer protocol), submitting forms by a web server and sending e-mail. (Denning, 1982). Information transferred in this way should be encrypted before transmission because unencrypted files travel as plain text, ready to be intercepted and interpreted by anyone. File encrypting adds a level of inconvenience, but in order to secure data, this inconvenience is unavoidable. Unfortunately, security decisions always involve a trade-off between security and convenience.

2.3.2 Verifying the integrity of files or messages

An important application of secure hashes is verification of message integrity. Determining whether any changes have been made to a message (or a file), for example, can be accomplished by comparing message digests calculated before, and after, transmission (or any other event). (Ramaratan Rathore, 2014). For this reason, most digital signature algorithms only confirm the authenticity of a hashed digest of the message to be "signed".

Verifying the authenticity of a hashed digest of the message is considered proof that the message itself is authentic. (Ramaratan Rathore, 2014)

2.3.3 Interactive Transmission

Interactive transmission involves two computers that have meaningful transmissions flowing in both directions. Interactive transmission, however, often involves spontaneous messages and must occur on both ends. In order to use any computer system over a network interactively, users must overcome two security exposures. First, users must authenticate themselves, and this exposes the authentication process to interception. Anyone sending out his or her password over the network is often sending that password out in plain text, which means anyone eavesdropping can pick up the password and username and use them. Stolen password and username combinations are the most common problem of interactive transmission. The other problem occurs while the user is using the system. The information being typed in is most likely going out as plain text, which can be intercepted. There are a few systems designed to limit the security risk in using a remote system interactively. An example is the Kerberos. (Denning, 1982)

2.4 Encryption

Encryption is the process that converts plain text into cipher text by using encryption algorithms with key that process performed on sender end and decryption is the reverse process of encryption performed on receiver end (Stallings, 2003). Encryption is the standard means of rendering a communication private. The sender enciphers each message before transmitting it to the receiver. Only the authorized receiver knows the appropriate deciphering function to apply to the received message to obtain the original message. An eavesdropper who hears the transmitted message only hears ñgarbageò (the cipher text), which makes no sense to him since he does not know how to decrypt it (Oded, 2004).

2.4.1 Encryption domains and co domains

A; denotes a finite set called the alphabet of definition. M; denotes a set called the message space. M consists of strings of symbols from an alphabet of definition. An element M is called a plaintext message or simply a plaintext. For example M may consist of binary strings, English text, computer code etc. C; denotes a set called the cipher text space. C consists of strings of symbols from an alphabet of definition, which may differ from the alphabet of definition for M. An element of C is called cipher text. (Oded, 2004).

2.4.2 Encryption and decryption transformations

K; denotes a set called the key space. An element of K is called a key. Each element e 2 K uniquely determines a bijection from M to C, denoted by Ee. Ee is called an encryption function or an encryption transformation. Note that Ee must be a bijection if the process is to be reversed and a unique plaintext message recovered for each distinct cipher text or each d 2 K, Dd denotes a bijection from C (i.e. Dd: C-! M). Dd is called a decryption function or decryption transformation. The process of applying the transformation Ee to a message m2 M is usually referred to as encrypting m or the encryption of m. The process of applying the transformation Dd to a cipher text c is usually referred to as decrypting c or the decryption of c. An encryption scheme consists of a set {Ee: e is element of K} of encryption transformations and a corresponding set {Dd: d is element of K} of decryption transformations with the property that for each e element of K there is a unique key D element of *K* such that Dd = Ee-1; that is Dd, (Ee(m)) = m for all m element of M. An encryption scheme is sometimes referred to as a cipher. The keys e and d in the preceding definition are referred to as a key pair and sometimes denoted by (*e,d*). Note that e and d could be the same. To construct and encryption scheme requires one to select a message space M. a cipher text space C, a key space K, a set of encryption transformations {Ee:e element of K}, and a corresponding set of decryption transformations{Dd:d element of K} (Oded, 2004).

2.4.3 Encryption Scheme

An encryption scheme may be used as follows for the purpose of achieving confidentiality; two parties Alice and Bob first secretly choose or secretly exchange a key pair ***(e; d)***. At a subsequent point in time, if Alice wishes to send a message *m element of* ***M*** to Bob, she computes ***c = Ee (m)*** and ***c*** transmits this to Bob. Upon receiving, Bob computes ***Dd (c) = m*** and hence recovers the original message ***m***. (Oded, 2004)

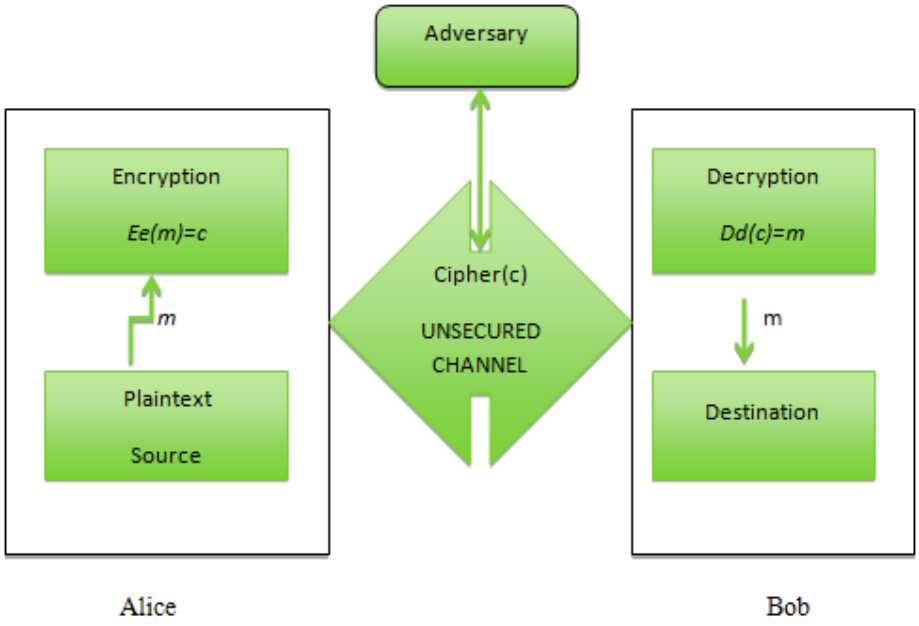


Fig 2.1 Provides a simple model of a two-party communication using encryption.

2.4.4 Communication Participants

An entity or party is someone or something, which sends, receives or manipulates information. Alice and Bob are entities in Fig 2.1 above. An entity may be a person, a computer terminal, etc. A sender is an entity in a two- party communication, which is the legitimate transmitter of information. In Fig 2.1, the sender is Alice. A receiver is an entity in a two-party communication, which is the intended recipient of information. In Fig 2.1, the receiver is Bob. An adversary is an entity in a two ï party communication which is neither the sender nor the receiver, and which tries to defeat the information security service provided between the sender and receiver. Various names are synonymous with adversary such as enemy, attacker, opponent, eavesdropper, intruder and interloper. An adversary will often attempt to play the role of either the legitimate sender or the legitimate receiver. A channel is a means of conveying information from one entity to another. An unsecured channel is one from which parties other than those for which the information is intended can reorder, delete, insert or read.

Encryption is the process of translating plain text into cipher text. (Markus Jacobsson, 1996). Human-readable information intended for transmission is plain text, whereas cipher text is the text that is actually transmitted. At the other end, decryption is the process of translating cipher text back into plain text. Encryption algorithm refers to the steps that a personal

computer takes to turn plain text into cipher text. A key is a piece of information, usually a number that allows the sender to encode a message only for the receiver. Another key also allows the receiver to decode messages sent to him or her.

It is reasonable to assume that at some point someone may intercept your transmissions. Whether you expect an interception or whether you just generally suspect that interceptions may occur, you should transmit your information in a format that is useless to any interceptors. A better system is one that allows you to send any message, even one you had not anticipated, to anyone without fear of interception. This is why an encryption system is so valuable; it allows any message to be transmitted that will be useless to anyone who intercepts it. Two types of encryption are private and public key encryption. (Stallings, 2003)

2.4.4.1 Symmetric Key Encryption

In secret key cryptography, a single key is used for both encryption and decryption. The receiver applies the same key to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called symmetric encryption. The term "private key" comes from the fact that the key used to encrypt and decrypt data must remain secure because anyone with access to it can read the coded messages. A sender encodes a message into cipher text using a key, and the receiver uses the same key to decode it (Ayushi, 2010).

Consider an encryption scheme consisting of the sets of encryption and decryption transformations {Ee:e element of K} and {D:d element of K}, respectively, where K is the key space. This is shown in figure 2.2 below. The encryption scheme is said to be symmetric-key if for each associated encryption/decryption key pair (e,d), it is computationally ñeasyò to determine d knowing only e, and to determine e from since e = d in most practical symmetric-key encryption schemes, the term symmetric key becomes appropriate.

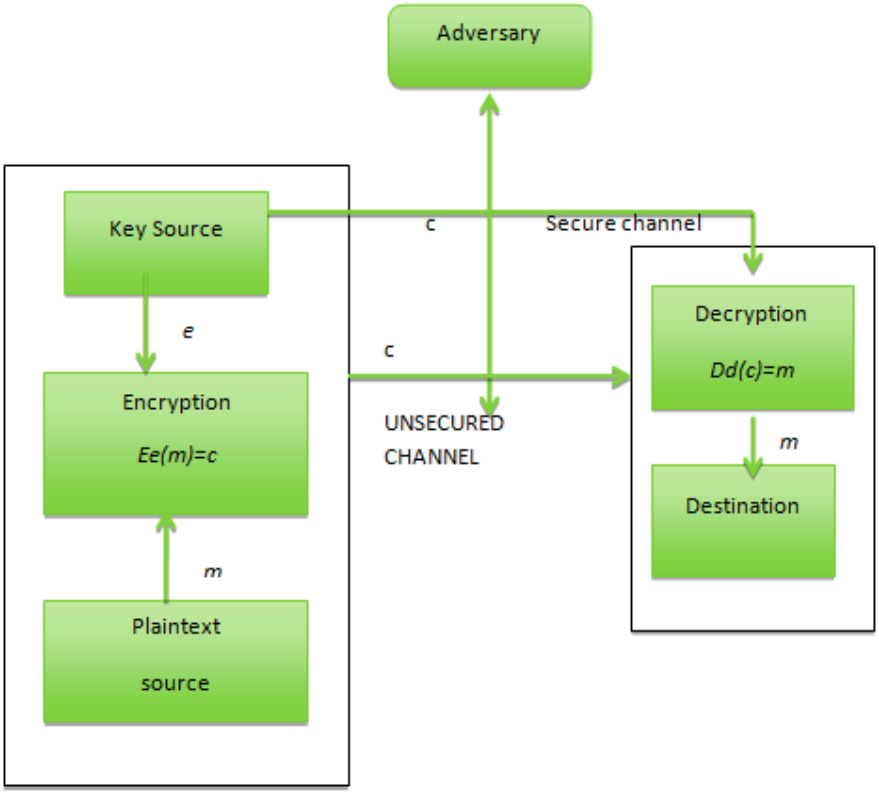


Fig 2.2- Two party communication using encryption

In a two party communication using encryption with a secure channel for key exchange, the decryption key **d** can be efficiently computed from the encryption key **e**. People can use this encryption method as either a "stream" cipher or a "block" cipher, depending on the amount of data being encrypted or decrypted at a time. According to Spillman (2005), a stream cipher encrypts data one character at a time as it is sent or received, while a block cipher processes fixed chunks of data. Common symmetric encryption algorithms include Data Encryption Standard (DES), Advanced Encryption Standard (AES), and International Data Encryption Algorithm (IDEA).

The drawbacks to private key systems, however, are twofold;

a. First, anyone who learns the method of encryption and gets the key, or a number or sequence of numbers or the sequences' equivalent of numbers that are used as a random input into the encrypted system, can break the key.

b. Second, keys must be exchanged before transmission with any recipient or potential recipient of your message. So, to exchange keys you need a secure method of

transmission, but essentially what you've done is create a need for another secure method of transmission. This problem is referred to as the key distribution problem. By using symmetric encryption approach. The symmetric encryption approach is divided in to two types one is block cipher symmetric cryptography technique and another is stream cipher symmetric cryptography but here block cipher type is used because its efficiency and security is good as compared to other. (krishna kumar, 2013)

2.4.4.2 Public Key Encryption

Asymmetric or public key, cryptography is, potentially, more secure than symmetric methods of encryption. This type of cryptography uses two keys, a "private" key and a "public key," to perform encryption and decryption. The use of two keys overcomes a major weakness in symmetric key cryptography, since a single key does not need to be securely managed among multiple users. To encrypt a message using a public key cipher, it is first converted to cipher text using the public key. The resulting cipher text can only be decrypted using the receiverôs private key. Secure communication now proceeds without any insecure exchanges of private information. (Jerome Burke, 2009). Asymmetric Key Encryption uses two mathematically associated keys: Public Key & Private Key for encryption. The public key is available to everyone but the data once encrypted by public key of any user can only be decrypted by private key of that particular user (Suyash Verma, 2012).

Consider an encryption scheme such that; Let *{Ee: e element of K}* be a set of encryption transformations, and let *{Dd: d element of K}* be the set of corresponding decryption transformations, where K is the key space. Consider any pair of associated encryption/decryption transformations *(Ee,Dd)* and suppose that each pair has the property that knowing *Ee* it is computationally infeasible, given a random cipher text *c* element of *C*, to find the message m element of M such that *Ee (m) = c.* This property implies that given e it is infeasible to determine the corresponding decryption key *d. (e* and *d* are simply means to describe the encryption and decryption functions, respectively.) *Ee* is being viewed here as a trapdoor one-way function with d being the trapdoor information necessary to compute the inverse function and hence allow decryption (Oded, 2004).

Under these assumptions, consider the two ï party communication between Alice and Bob illustrated in Fig 2.3 below. Bob selects the key pair *(e,d).* Bob sends the encryption key e (called the public key) to Alice over any channel but keeps the decryption key *d* (called the

private key) secure and secret. Alice may subsequently send a message m to Bob by applying the encryption transformation determined by Bobôs public key to get c = *Ee(m)*. Bob decrypts the cipher text *c*by applying the inverse transformation *Dd* uniquely determined by *d* (Oded, 2004). In asymmetric cryptography, a public key is freely available to everyone and used to encrypt messages before sending them. A different, private key remains with the receiver of cipher text messages, who uses it to decrypt them. Algorithms that use public key encryption methods include RSA and Diffie-Hellman. (Amalia R. Miller, 2010).

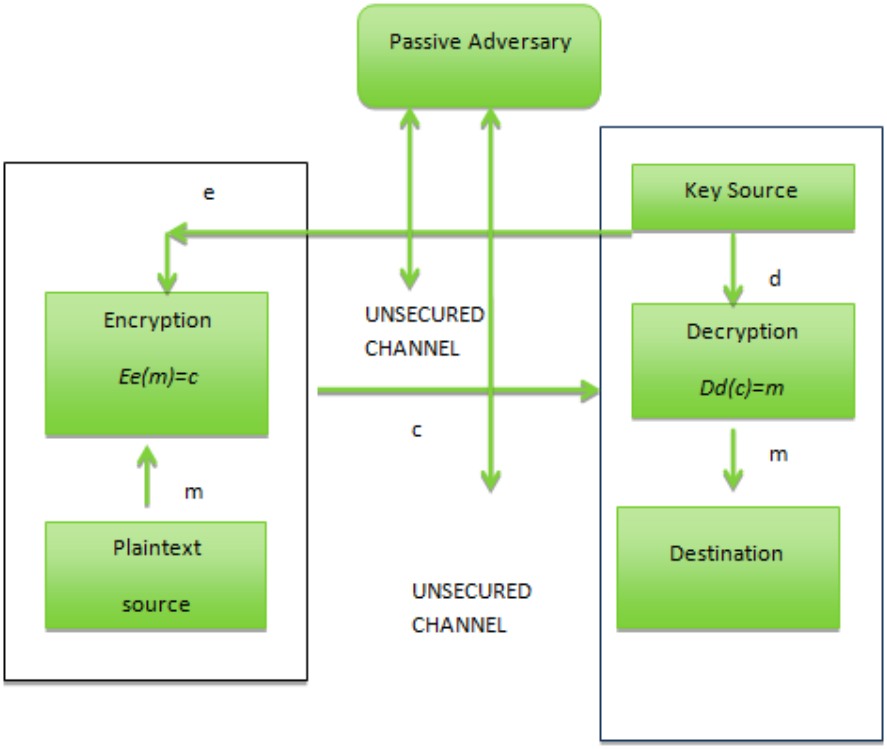


Fig 2.3: Encryption using public-key techniques

2.4.4.3 Digital Signature

An important application for public key cryptography is "digital signature", which can be used to verify the integrity of data or the authenticity of the sender of data. In this case, the private key is used to "sign" a message, while the corresponding public key is used to verify a "signed" message. Public key cryptography offers the benefits of confidential transmissions and digital signature in an open network environment in which parties do not know one another in advance (Gary Locke, 2009).

This cryptographic primitive is fundamental in authentication, authorization and non-

repudiation. The purpose of a digital signature is to provide a means from an entity to bind its identity to a piece of information. The process of signing entails transforming the message and some secret information held by the entity into a tag called a signature (Denning, 1982). The Set-up for a digital Signature is described below: *M* is the set of messages, which can be signed. *S* is a set of elements called signatures, possibly binary strings of a fixed length. *SA* is a transformation from the message set *M* to the signature set *S*, and is called a signing transformation for entity A. The transformation SA is kept secret by A and will be used to create signatures for messages from *M*. *VA* is a transformation from the set $M \times$ to the set {true, false}. *VA* is called a verification transformation for Aôs signature, is publicly known, and is used by other entities to verify signatures created by *A*. The transformations *SA* and *VA* provide a digital signature scheme for *A*. (Menezes & Van Oorschot, 1996)

2.4.4.4 Hash Functions

A hash function is any algorithm that maps data of a variable length to data of a fixed length. The values returned by a hash function are called hash values, hash codes, hash sums, checksums or simply hashes. Hash functions are primarily used to generate fixed-length output data that acts as a shortened reference to the original data. This is useful when the output data is too cumbersome to use in its entirety (Menezes & Van Oorschot, 1996).

One practical use is a data structure called a hash table where the data is stored associatively. Searching for a person's name in a list is slow, but the hashed value can be used to store a reference to the original data and retrieve constant time (barring collisions). Another use is in cryptography, the science of encoding and safeguarding data. It is easy to generate hash values from input data and easy to verify that the data matches the hash, but hard to 'fake' a hash value to hide malicious data. This is the principle behind the Pretty Good Privacy algorithm for data validation (Menezes & Van Oorschot, 1996).

2.5 Classes of Security Attacks

1. Passive attack is one where the adversary only monitors the communication channel. A passive attacker only threatens confidentiality of data. This is also known as the unauthorized access.
2. Active attack is one where the adversary attempts to delete, add or in some other way alter the transmission on the channel. An active attacker threatens data integrity and authentication as well as confidentiality. This is also known as destruction attack.

3. Editing is where the attacker is able to obtain and change any information stored in a computer system. With editing the attacker can change code blocks of a program or content of an important file. As a result, it could be impossible for the user to execute important files and programs again.
4. Reproducing is where the attacker sends a fake message to a computer user. The user thinks that the message sender is a trusted one. But the message or its attachments may contain malicious content.

2.6 Review of Existing Encryption Methods.

The principal goal of designing any encryption algorithm is to hide the original message and send the non-readable text message to the receiver so that secret message communication can take place over the web. The strength of an encryption algorithm depends on the difficulty of cracking the original message. A number of symmetric key encryption algorithms like DES, Triple DES; AES has been developed to provide greater security affects one over the other (Monika. A, 2012). Encryption is a method for a user to securely share data over an insecure network or storage site. Before the advent of public key cryptography, a widely held view was that for two users to communicate data confidentially they would need to a priori establish a mutually held secret key. While this might be acceptable for some small or tightly knit organizations, such a solution was clearly infeasible for larger networks such as todayôs Internet consisting of billions of users (Dan Boneh, 2006).

Many cryptographic algorithms have already been proposed and implemented to provide security to the user that his/her message would remain safe at the time of communication over the web. But now with the increase of attacks in the society such cryptographic algorithms are longer safe. Below are some of the existing algorithms.

2.6.1 Data Encrypt Standard (DES)

DES (the Data Encryption Standard) is a symmetric block cipher developed by IBM. The algorithm uses a 56-bit key to encipher/decipher a 64-bit block of data. The key is always presented as a 64-bit block, every 8th bit of which is ignored. However, it is usual to set each 8th bit so that each group of 8 bits has an odd number of bits set to 1 (Hyubgun Lee, 2009).

DES is a 64-bit block cipher under 56-bit key. The algorithm processes with an initial permutation, sixteen rounds block cipher and a final permutation. DES application is very

popular in commercial, military and other domains in the last decades (Hyubgun Lee, 2009). DES algorithm is vulnerable to linear cryptanalysis attacks. By such an attack, the algorithm in its sixteen rounds can be broken. The vulnerability raises a notable risk when encrypting bulk data that may be predictable with keys that are constant (Elbaz, 2002).

2.6.2 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. It is based on the Rijndael cipher.AES is a variant of Rijndael, which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification *per se* is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits. A simpler way to view the AES function order is: Scramble each byte (Sub byte), Scramble each row (Shift rows), Scramble each column (Shift column) and Encrypt (Add round key)

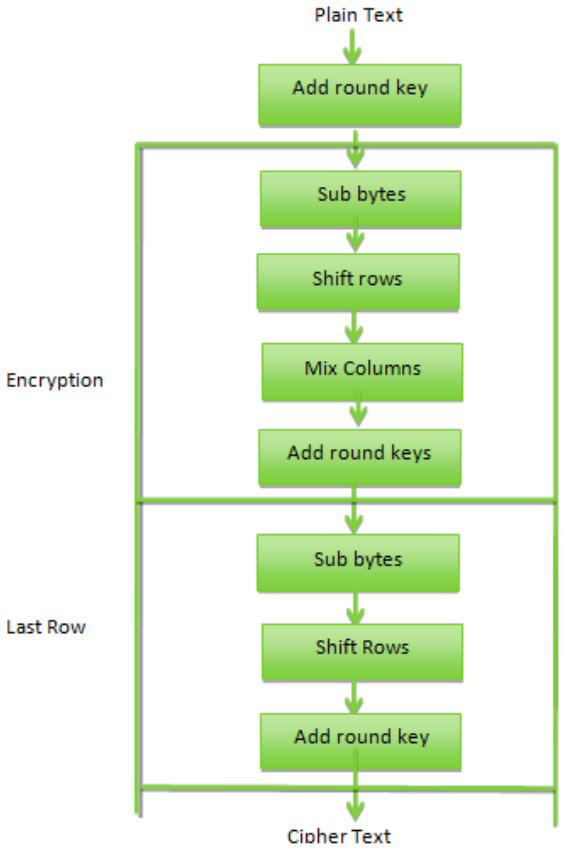


Fig 2.4: AES Encryption.

2.6.3 Diffie-Hellman Key Exchange

This is a specific method of exchanging cryptographic keys. It is one of the earliest examples of key exchange implemented within the field of cryptography. The Diffie-Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher (Mahalanobis, 2005). Diffieï Hellman establishes a shared secret that can be used for secret communications while exchanging data over a public network. The best-known cryptographic problem is that of privacy (Hellman, 1976), preventing the unauthorized extraction of information from communications over an insecure channel. In order to use cryptography to insure privacy, however, it is currently necessary for the communicating parties to share a key, which is known to no one else. Sending the key in advance over some secure channel such as private courier or registered mail does this. A private conversation between two people with no prior acquaintance is a common occurrence in business, however, and it is unrealistic to expect initial business contacts to be postponed long enough for keys to be transmitted by physical means (Palmgren, 2006).

The Actual Process

Diffie ï Hellman is a method to securely exchange the keys that encrypt data. This is accomplished by creating a ñshared secretò (Key Encryption Key) between two devices. The shared secret then encrypts the symmetric key for secure transmittal. The symmetric key is sometimes called a traffic encryption key or data encryption key. Therefore, the KEK provides for secure delivery of the TEK, while the TEK provides for secure delivery of the data itself.

Step 1

The process begins when each side of the communication generates a private key. Each side then generates a public key, which is a derivative of the private key. The two systems then exchange their public keys. Each side of the communication now has their own private key and the other systems public key. Noting that the public key is a derivative of the private key is important; the two keys are mathematically linked. The certificate authority may be used to certify that the public key is indeed coming from the source you think it is. This is to prevent

ñman in the middleò potentially intercepts encrypted traffic, decrypts it, copies or modifies it, re-encrypts it with the bogus key and forwards it on to its destination. If successful, the parties on each end would have no idea that there is an unauthorized intermediary.

Step 2

Once the key exchange is complete, the process continues. The Diffie Hellman protocol generates ñshared keysò i.e. identical cryptographic keys shared by each side of the communication. By running the mathematical operation against your own private key and the otherôs side public key, you generate a value. When the distant ends run the same operation against your public key and they own private key, they also generate a value. The two values generated are identical. They are the ñshared secretò that can encrypt information between systems.

Step 3

The shared secret encrypts a symmetric key for one of the asymmetric algorithms transmits it securely and the distant end decrypts it with the shared secret. Because the symmetric key is short value (256 bits) as compared to bulk data, the shared secret can encrypt and decrypt it very quickly.

Step 4 – Encrypted Data Transmission

Once secure exchange of the symmetric key is complete (note, that passing of the Diffie Hellman key is the whole point of the Diffie-Hellman operation), data encryption and secure communication can occur. The longer a symmetric key is in use, the easier it is to perform a successful cryptanalytic attack against it. Both sides of the communication still have the shared key and it can be used to encrypt future keys at any time and any frequency desired.

2.6.4 RSA – Public Key Protocol

Below are the algorithms performed to get the private and the public key. To convert the data, mathematical calculations and algorithms are used to create an encryption. One of the simplest forms of encryptions is alphabet substitution, where each alphabet would be systematically or randomly scrambled in a certain order to create a key. An example of would be each alphabet is represented in the encrypted data as the sixth alphabet that follows after it, for instance, the alphabet "**F**" to represent the alphabet "**A**". The party who has the key would be able to easily decrypt the information, but people who do not have key can also decrypt

the information with good mathematical knowledge and persistence.

The challenge with this protocol is that the factorials used to generate these keys are very large numbers and difficult to calculate. Modern computers are currently unable to factor this algorithm and hence the protocol is rarely used.

The RSA algorithm can be attacked by brute force attack, mathematical attacks, cryptanalysis attacks and timing attacks. When using RSA for signature, an attacker can cause a person to sign a message that he/she would not willfully sign may use the blinding features of the algorithms. Applying a hash function on the message prior to signing can effectively solve this problem. This will allow the blinding feature but will make the blinding ï based attack feasible.

In 1996, (et, 1996)presented an attack on RSA doing faulty calculations. By injecting random faults into the calculations of RSA, they were able to regenerate the private key from the knowledge of faulty signatures. Similar attacks have been shown subsequently. For example, (Eric Brier, 2006) provide a fault-based attack on the RSA modulus operation. The assumption is that an attacker is able to input random bits in the modulus.

2.7 State of the art

2.7.1 RSA (Public-Key Encryption)

RSA is a cryptosystem, which is known as one of the first practicable public key cryptosystem and is yet widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key, which is kept secret. This asymmetry is based on the practical difficulty of factoring the product of two prime numbers i.e. the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the algorithm in 1977. RSA is used to provide secrecy and digital signatures and its security is based on the intractability of integer factorization. The RSA algorithms involve three steps: key generation, encryption and decryption.

2.7.2 PGP (Pretty Good Privacy)

Pretty Good Privacy (PGP) is a data encryption and decryption computer program that provides cryptographic privacy and authentication for data communication. PGP is often used for signing, encrypting and decrypting texts, e-mails, files, directories and whole disk partitions and to increase the security of e-mail communications. It was created by Phil Zimmerman in 1991. PGP encryption uses a serial combination of hashing, data compression, symmetric-key cryptography and public-key cryptography. Each public key is bound to a user name and/or an e-mail address. PGP combines symmetric-key encryption and public-key encryption. The message is encrypted using a symmetric encryption algorithm, which requires a symmetric key. Each symmetric key is used only once and is called a session key. The message and its session key are sent to the receiver. The session key must be sent to the receiver so they know how to decrypt the message. However, in order to protect the session key during transmission; it is encrypted with receiverôs public key. Only the private key belonging to the receiver can decrypt the session key (Jonathan, 2000).

PGP supports message authentication and integrity checking. The latter is used to detect whether a message has been altered since it was completed (message integrity) and the former to determine whether it was actually sent by the person or entity claimed to be the sender (digital signature). The cryptographic security of PGP encryption depends on the assumption that the algorithms used are unbreakable by direct cryptanalysis with current equipment and techniques. RSA algorithm is used to encrypt session keys. RSAôs security depends upon the one-way function nature of mathematical integer factoring (Randall, 1999).The symmetric key algorithm used in PGP is IDEA.

2.8 Differences from other Encryption Algorithms and Proposed Algorithm.

The proposed algorithm has a similar concept as other symmetric encryption whereby similar keys are used for encryption and decryption. The proposed algorithm with AES encryption algorithm using a guard to generate keys for encryption and decryption; this should be secure enough so that an adversary or malicious attacker does not access it. Proposed technique uses a common key between sender and receiver, which is known as private key. Basically private key concept is the symmetric key concept, where plain text is converted into encrypted text known as cipher text using private key and cipher text is decrypted by same private key into plain text. (krishna kumar, 2013)

In this research, the security is focused on applying sound cryptographic fundamentals in the exchange of the secret keys between the client and the server, thus giving the benefit that only the people who should have access to the data are actually the ones who get the keys used for encryption and decryption of the data.

In this research, we developed a model that ensures:
How encryption can be applied in the secure transmission of Examination files and also the encryption mechanisms and algorithms implemented within the framework can be easily maintained between the client (browser) and the Server.

2.9 Direction in which research is heading

Despite the enormous strength of RSA and other modern ciphers, cryptanalysts are still able to play a valuable role in intelligence gathering. The strength of an algorithm is based on two things: how good the mathematics is, and how long the key is. A sure way of breaking an algorithm is to try every possible key (Aydin, 2009). Modern algorithms have a key so long that this is impossible; even if you built a computer out of all the silicon atoms on the planet and ran it for millions of years, you couldn't do it. So cryptographers look for shortcuts. If the mathematics is weak, maybe there's a way to find the key faster: "breaking" the algorithm.

Only a small component of the information flowing around the world is securely encrypted, and the remainder is poorly encrypted, or not encrypted at all. This is because the number of Internet users is rapidly increasing, and yet few of these people take adequate precautions to ensure their privacy. This means that national security organizations, law enforcers or anybody with a curious mind can access more information than they can cope with.

The RSA cipher is not necessarily an effectively unbreakable encryption system. RSA is presumably breakable if the number factorization problem is simplified. Quantum computing holds the promise of achieving this. The only obstacle is the infancy of quantum computing and the cost of the required resources. The really crucial question is whether quantum cryptography will arrive in time to save us from the threat of quantum computers, or whether there will be a privacy gap, a period between the development of quantum computers and the advent of quantum cryptography. The most effective way to use cryptography is within the context of a sound security policy, which creates a framework for effective management of all security-related systems and keeps users informed as to their individual roles and

responsibilities (Stephen F. Bisbee, 1997).

2.10. Developed Model for Secure transmission of Examination Data

This research develops a model which encrypts and uploads text files and then confirms the vulnerability of the encrypted files. Uploaded are examination texts on the prototype using dynamic symmetric keys, the plaintext file encrypts and the cipher text file stored in the MySQL database. With the increase of geographically distributed campuses within Universities, there is need to secure the data that is shared amongst them. Examination data is very sensitive and access to it by unauthorized personnel might result to stakeholders lose confidence in the Examination process.

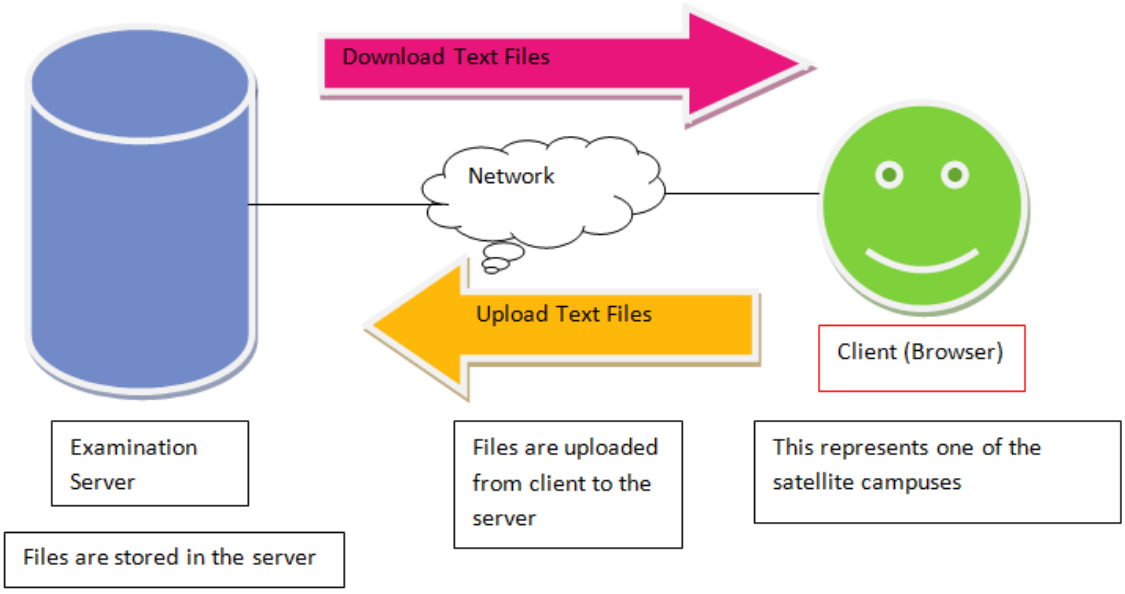


Fig 2.5: Abstract view of the Examination management system

To address some of the important security challenges, which are discussed above, the research proposes an algorithm for secure channel of transmission of examination data which uses AES-based encryption with dynamic key generator because of the resistance against some attacks, speed and code compactness on a wide range of platforms and design simplicity. The core objective is to strive for the best combination amongst security, simplicity and efficiency in cryptography. In order to cater for the different computing machines involved in this application environment, the proposed model has two implementation:

1. Client (Browser) - This implementation is targeted for the standalone computers used to upload the examination files.
2. Server (Main examination) ï This implementation is targeted for the server machine such as a desktop PC, where the examination files will be stored.

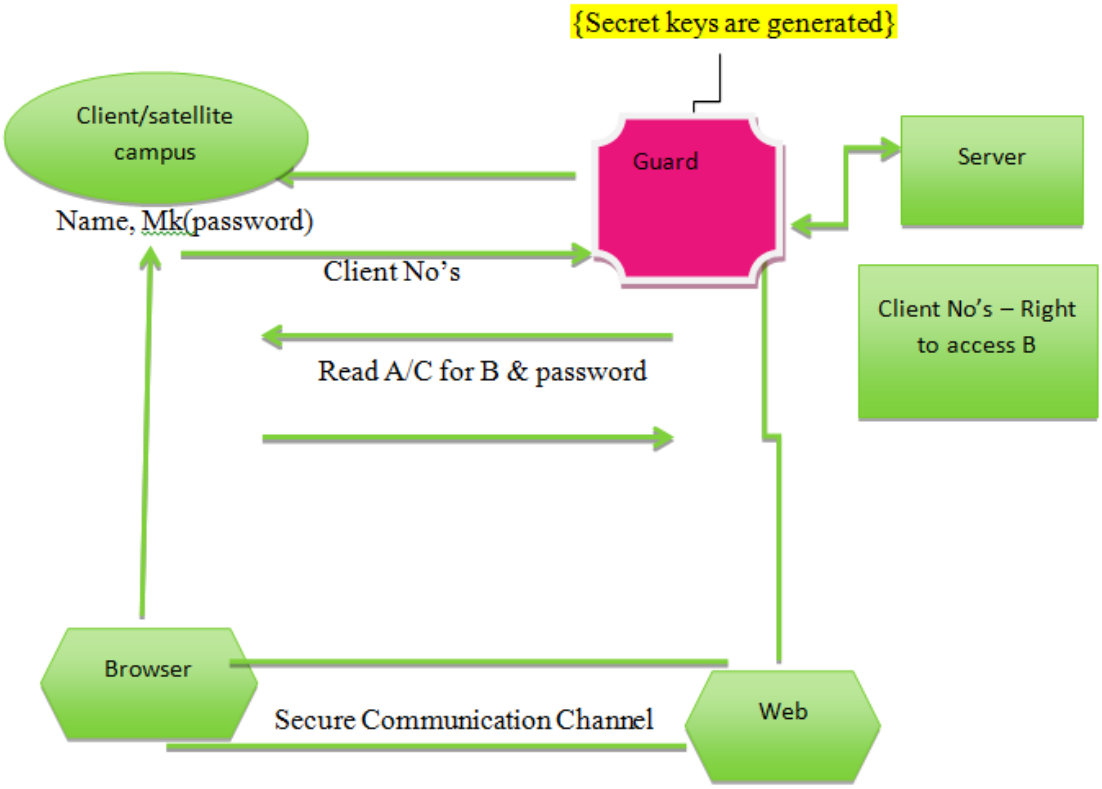


Fig 2.6: Conceptual Model for Secure transmission of Examination Data

Client is one of the satellite campuses who use the online examination portal to upload data. **Web** is where the portal is accessed. **Guard** acts as a shield to protect unauthorized people from accessing the server. This includes, Password and access controls. Secret keys are generated dynamically to encrypt and decrypt uploaded files. **Secure Communication Channel**ï symmetric encryption is used to encrypt and decrypt the uploaded text file.

CHAPTER THREE
RESEARCH METHODOLOGY

3.1 Introduction

This chapter presents the research methods that were followed and the chapter starts by describing the activities that were carried out to achieve the overall objectives outlined above. The research undertakes the following phases: Survey literature, Design the model of the guard, Simulate examination management environment, Run some tests and evaluate the model, Discuss results and Data Collection .

The research employed a literature review survey as a means to understand how encryption can be implemented in the examination environment as documented. Basically no research work ever stands on its own. Research is for a large extent the recognition of the validity of previous ideas and its applicability to different settings. This survey goes further to justify and give a clear glimpse of types of encryption currently used in modern systems and their importance in ensuring security of data at all times. Different encryption algorithms were reviewed and AES was adopted and improved with the implementation of a guard that dynamically generates secrets keys which are used to encrypt and decrypt files as there are being uploaded. It also intends to investigate and unearth original information to understand the security issue at hand. Finally the problem being examined has a significant social, economic impact on educational sector especially universities where the campuses are geographically distributed.

The guard was designed using AES encryption algorithm with a dynamic key generator. The secret keys generated were used to encrypt and decrypt data in transmission. After the design of the guard, a prototype was developed and a simulation of the examination management system was done. A simple interface was designed and developed using java web system to provide an easy and fast user experience for computer users. An encryption algorithm was used to encrypt plaintext into cipher text for secure transmission of examination text files. The user can upload text files on the application and using secret keys the text files are encrypted and stored in a MySQL database. The application then uses the secret key and decrypts the cipher text into plaintext in the case of decryption. A test was conducted for evaluation purposes; the application was tested on a java web system which is built in net beans. The application was then run randomly by computer science students who are known

for trying to access examination materials from the examination portal. The results were then discussed after analyzing the security logs of the number of attempted users in the system.

The study utilized both secondary and primary data in generating new depth on the subject. The primary data was collected through observations of different examinations stakeholders in different satellite campus to get a better insight on how security issues dealing with examinations are handled to ensure that the process is not tampered with. The security logs collected were also used to analyze the data.Wapiti was used to analyse the logs for attacks for a period of two weeks. Wapiti allows one to audit the security of web applications. It performs "black-box" scans, i.e. it does not study the source code of the application but will scan the web pages of the deployed webapp, looking for scripts and forms where it can inject data.

3.2 Design and Development of the guard

Encryption is the process that converts plain text into cipher text by using encryption algorithms with key that process performed on sender end and decryption is the reverse process of encryption performed on receiver end (Stallings, 2003). Encryption is the standard means of rendering a communication private. The sender enciphers each message before transmitting it to the receiver. Only the authorized receiver knows the appropriate deciphering function to apply to the received message to obtain the original message. An eavesdropper who hears the transmitted message only hears ñgarbageò (the cipher text), which makes no sense to him since he does not know how to decrypt it (Oded, 2004).

A key is a piece of information, usually a number that allows the sender to encode a message only for the receiver. Another key also allows the receiver to decode messages sent to him or her. The reason why an encryption system is so valuable is that, it allows any message to be transmitted that will be useless to anyone who intercepts it. Shown below, in figure 3.1, is the current AES encryption algorithm before incorporating a key generator.

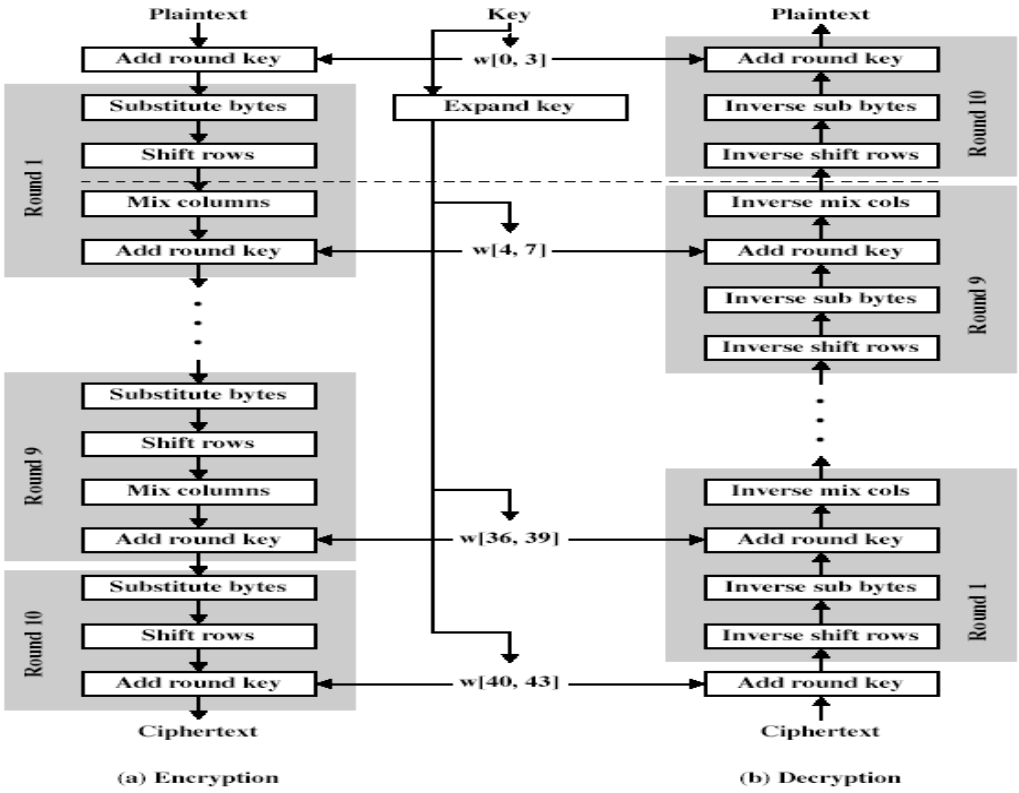
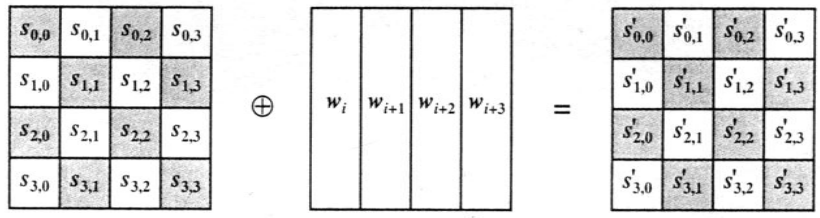


Fig: 3.1 AES Encryption Algorithm (Source: Menezes et al. (1996))

The algorithm works through the following steps (Menezes et al., 1996):

Step 1 - AddRoundKey

- Each round uses four different words from the expanded key array.
- Each column in the state matrix is XORed with a different word.
- The heart of the encryption. All other functionsô properties are permanent and known to all.



(b) Add Round Key Transformation

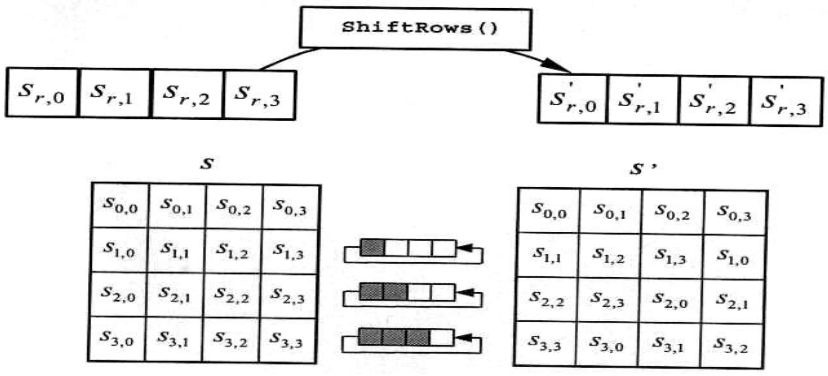
$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \\ b_4' \\ b_5' \\ b_6' \\ b_7' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Step 2 - Shift Rows

- a circular byte shift in each row

1st row is unchanged, 2nd row does 1 byte circular shift to left, 3rd row does 2 byte circular shift to left and 4th row does 3 byte circular shift to left.

- decrypt does shifts to right
- since state is processed by columns, this step permutes bytes between the columns
- Rows 2-4 in the state matrix are left shifted by different offsets of 1-3 bytes respectively.
- Strong diffusion effect. Separation of each four, originally consecutive, bytes.



Step 3 - Mix Columns

- each column is processed separately
- each byte is replaced by a value dependent on all 4 bytes in the column
- effectively a matrix multiplication in GF(2^8) using prime poly m(x) =$x^8+x^4+x^3+x+1$
- A transformation which operates on individual columns ï 32 bits/4 bytes.
- Each column is treated as a 3 degree polynomial over GF(2^3)

Using AES encryption algorithm, a key generator was designed which dynamically generates secret keys that are used to encrypt and decrypt the uploaded and downloaded text files. This was done by the block cipher which encrypts fixed-size block of data and is more secure than the stream cipher.

3.3 A Model of the AES-Based Encryption with Key Generator

The Fig 3.2 below show a model of AES based encryption with key generator for improved security. In the application, the data is separated into blocks first. Later, using random permutation production produces keys. At last, using XOR structure, which is also called as bit-level encryption, does key production and encrypting process. After this step, the encrypted text can be sent to the receiver. The receiver can read the data message by decrypting it with the same application.

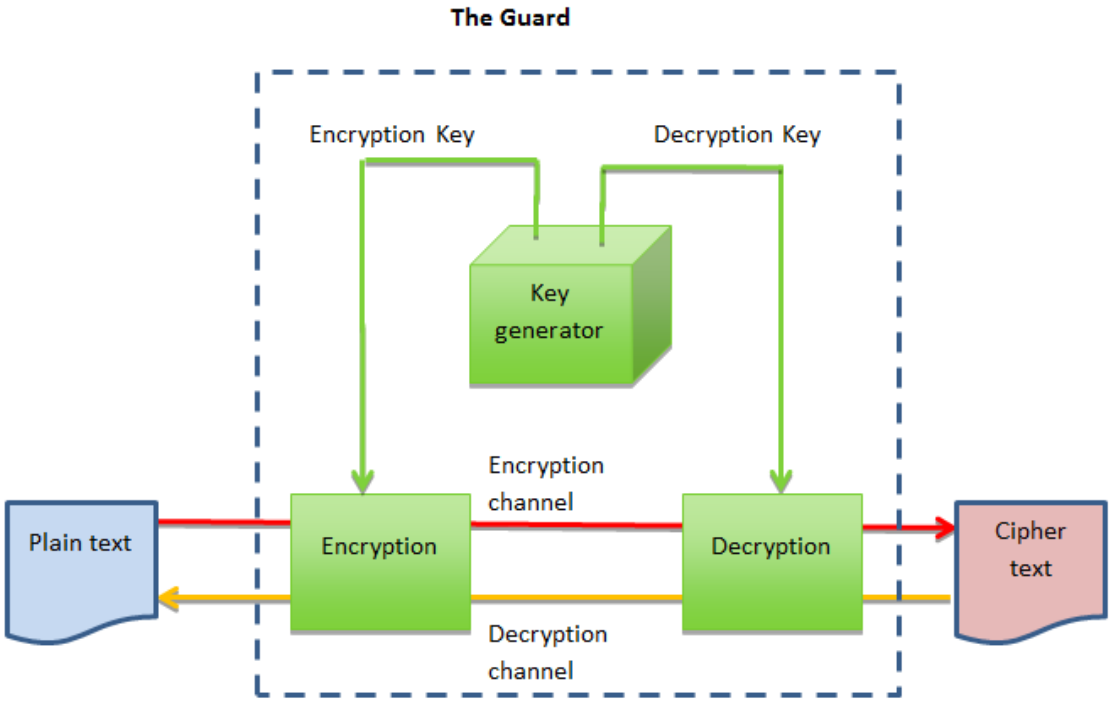


Fig 3.2 Model of the improved AES-Based Encryption with Key Generator

3.3.1 How the model works

Generate the encryption key: generates a java security key that is a valid AES algorithm encryption key. The generator object must then be initialized with a java security Secure Random that grants secure random numbers, as we need the random number to be

cryptographically strong.

Encrypt the message: Encryption of a message involves: Building the Cipher object that will be responsible of doing the encryption, converting the message to a format suitable for the Cipher and finally, converting the encrypted message to a format that will be easy to read on screen

Decrypt the message: The steps for decrypting the message are basically doing the reverse steps in the reverse order as the encryption i.e. Obtain a Cipher, same as for the encryption, secondly, decode the cipher message that we need to pass in to the decrypter i.e. same string that we got out from the encrypter. Finally, decrypt the message using the Cipher. Note: the only data in common between the encryption and the decryption methods is the key.

3.3.2 Steps for the function of the key generator module

The following are the steps on how the **key generator** works:

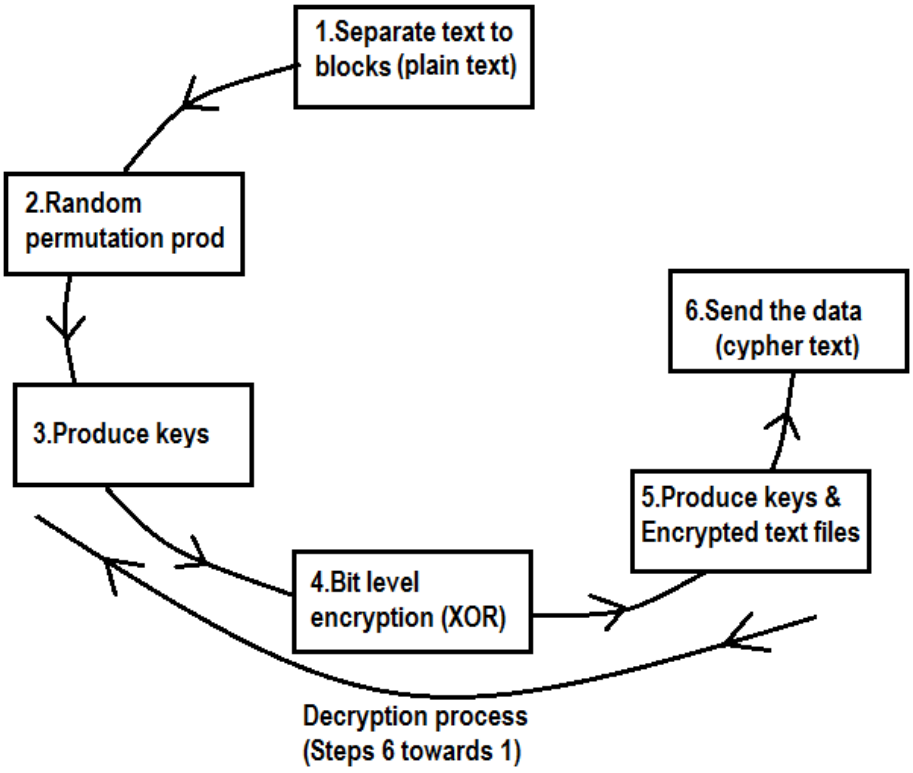


Fig 3.3 Process of How Guard (Key Generator) works

From the diagram above, the steps are explained as follows:

Separating into Blocks: In the first step of encryption, the original text is separated into different number of blocks according to character count. Character count is set to a definite number, which can be divided into five. Adding space characters to the original text can do this. This is also called padding.

Random Permutation Production: After separating the text into blocks, random numbers are produced for each block. The produced numbers are used to change positions of each character in the blocks. After changing positions of the characters, an encrypted text, which was created with the permutation method, is obtained. In the permutation method, properties of the text characters are protected. But their positions and orders are changed. For general permutation encoders, the encryption key is as much as the length of the permutation and the decryption key is as much as the length of the inverse permutation. This design principle is known as a substitution-permutation network and is fast in both software and hardware.

Producing Keys: In this step, random numbers between 0 and 9 are produced according to character count of the encrypted text.

Bit-Level Encryption (XOR): Each character of the text, which was encrypted with the permutation method, is encrypted at a bit level with a random key number. AES operates on a 4x4 column-major order matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field. The key size for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the cipher text.

Producing Keys and Encrypted Text Files: Two plain text files (.txt) are created for the produced key and the encrypted data. The created files can be saved into any directory. It is also important to protect key file from malicious people. After producing the key, numbers that were produced with random permutation method are added to the key.

Sending the Data: The encrypted data is sent to the receiver in this process.

Decrypting the Message: The received message is decrypted in this process. The key that is used for encryption/decryption and its parts can be described in the following way.

3.3.3 Pseudo code of the Key Generator

- The input data block is broken into a 4*4 byte array (128-bit key)
- The initial sub key (derived from the cipher key via key schedule [8]) is XORôd to the byte array by an *AddRoundKey()* operation(Nb = block size)
- For each encryption ñroundò/iteration to n-1 (128-bit, n=10; 192-bit, n=12; 256-bit,

n=14):

- Each array byte is substituted using a subbytes() S-box[7] operation.
- *ShiftRows ()* cyclic shift operation is done on the last three rows of the byte array.
- The array columns are then modified by a *MixColumns()* operation.
- *AddRoundKey*() is used again for the next round before *SubBytes().*
- In the nth round, all operations but *MixColumns()* are performed.
- Decryption is near-reverse, by order of operations (not structure), using Inv*() functions:
    - InvShiftRows()
    - InvSubBytes()
    - AddRoundKey()
    - InvMixColumns()
- The primary encryption module calling every other function:

- Cipher (byte in [4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
- Begin
- Bytestate[4,Nb]
- State = in
- AddRoundKey(state, w[0, Nb-1])
- For round = 1 step 1 to Nrï 1

SubBytes(state)

ShiftRows(state)

MixColumns(state)

AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])

- end for
- SubBytes(state)
- ShiftRows(state)
- AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
- out = state
- end
- Key expansion to generate the key schedule
- The primary decryption module calling every other function:
- InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
- begin
- byte state[4,Nb]

- state = in
- AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
- for round = Nr-1 step -1 downto 1

InvShiftRows(state)

InvSubBytes(state)

AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])

InvMixColumns(state)

- end for
- InvShiftRows(state)
- InvSubBytes(state)
- AddRoundKey(state, w[0, Nb-1])
- out = state

end

3.3.4 Development of the Prototype

Using the waterfall methodology a prototype was designed using the following platforms:

i) MySQL (Structured Query Language) database ï This database served as the backend database for the storage of data. The data stored included the relevant security codes/keys. Other data available was based on the need and preference of the user.

ii) Java web system where server being used is glassfish server 3.2. Server side is built using java 7, jsf (2.0) and interface (client side) is built on html5, css3, jquery 1.7, JavaScript and prime faces.

iii) For logs, xml files and a java api from the log4j library was used.

The prototype implemented the simulation of an examination management environment as shown in figure 3.4 below.

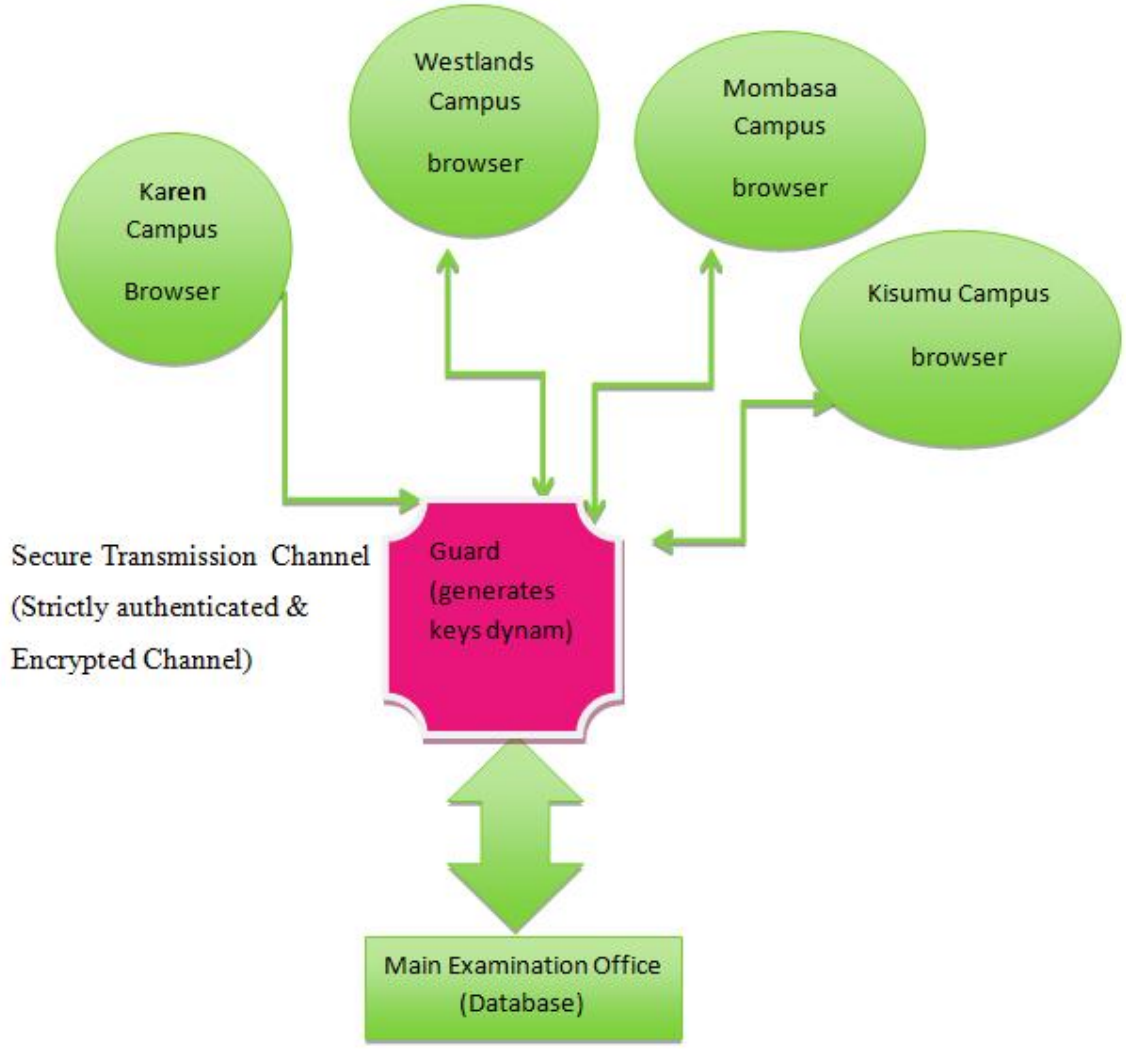


Fig 3.4: Examination Management Model

3.3.5 Implementation Tools

3.3.5.1 The programming Language

This Application is built on the Microsoft Windows 2007 platform using Java web system and XAMMP. The application is designed in the form of an interface (client side) ï server model. Server being used is glassfish server 3.2, MySQL server for database. The server side is built using java 7 and jsf(2.0). The client side is built on html5, css3, jQuery 1.7, JavaScript and prime faces. In the application, the server generates AES (Advanced Encryption Standard) encryption/decryption keys and keeps an encrypted log file of the encrypted text files. Logs file were also implemented on the application.

3.3.5.2 The Database Management Software

Being a web application, the system sits on a database. The chosen Database Management software is MySQL because it is a fast multi-threaded and multi user robust database management system. Using MySQL ensures that all the system information can be managed from a single database file with separate tables. MySQL is also chosen because of its ease of comp ability with java and its availability as open software.

3.3.5.3 The Web Server

The web application needs to run on a web server in order to be able to be availed throughout the organization. The chosen web server was glassfish server 3.2.The client side is built on html5, css3, jQuery 1.7, JavaScript and prime faces. XAMMP was also used in the implementation.

3.3.5.4 The GUI

The GUI is developed using html5, css3, jQuery 1.7, JavaScript and prime faces which are widely used for designing, coding and developing of web pages, websites and web applications.

3.3.6 The final prototype

An online examination management application was developed. A simple interface was designed and developed using java web system to provide an easy and fast user experience for computer users. An encryption algorithm was used to encrypt plaintext into cipher text for secure transmission of examination text files. The user can upload text files on the application and using secret keys the text files are encrypted and stored in a MySQL database. The application then uses the secret key and decrypts the cipher text into plaintext in the case of decryption. The proposed approach can be divided into two phases:

a) Enrollment Phase.
b) Login Phase.

3.3.6.1 Enrollment Phase

If a user visits a website first time, the user must register at the server. Assume that the connection between the user and the server in the registration is in a secure channel. Firstly the user details such as username, userôs phone Number, email id, campus and unit taught are

saved in the server. Server generates a unique key and user name and stores along with other details in the database at the time of registration at the database of the secure website.

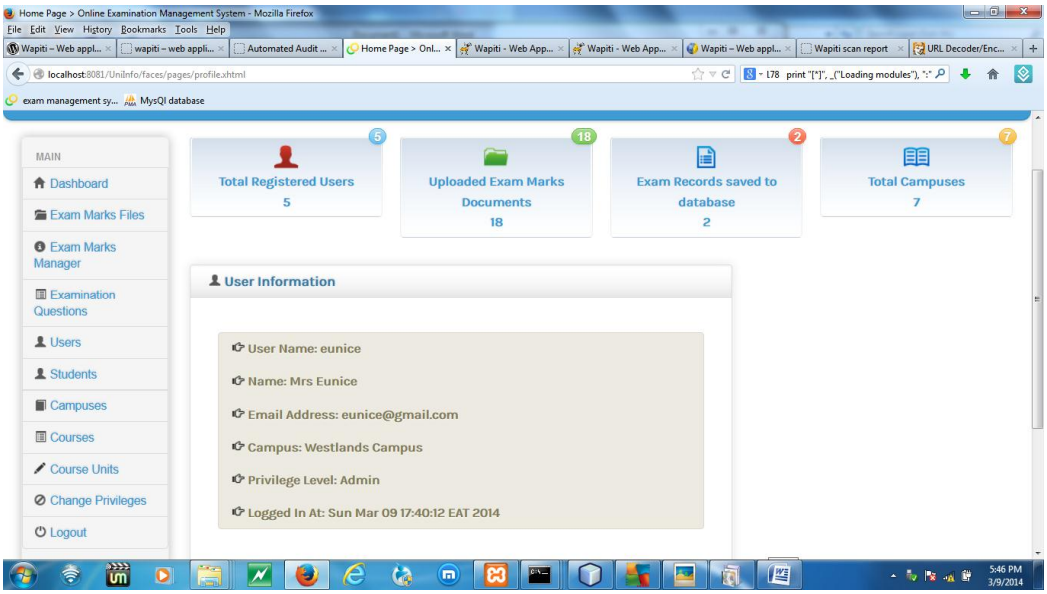


Fig 3.5: Enrolment Screen

3.3.6.2 Login Phase

When the user logs in, user inputs his/her user name and password. Once the server verifies that the user is genuine, it allows access to the dashboard whereby the user can upload a document.

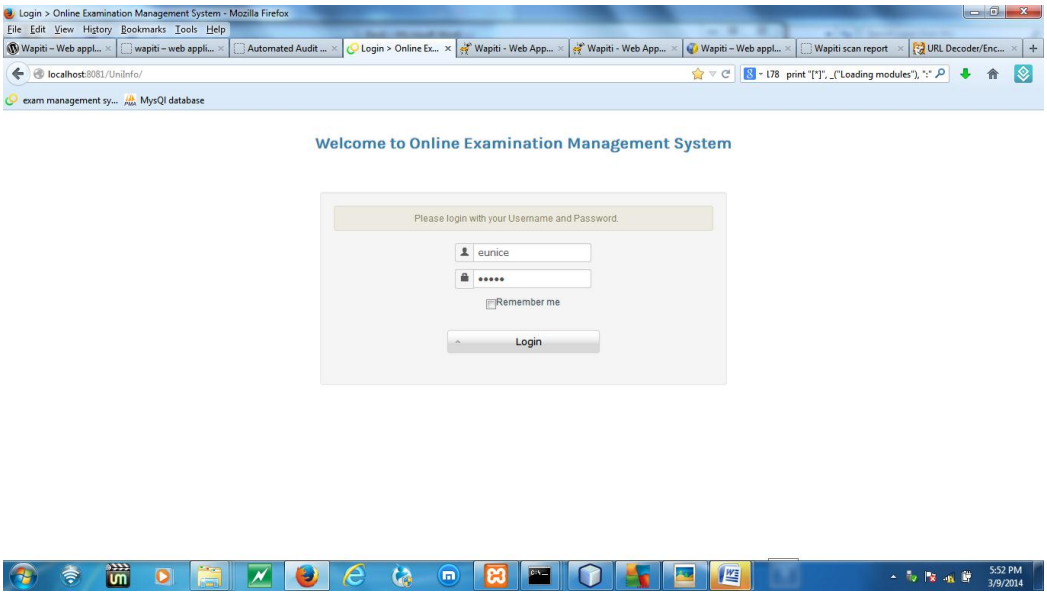


Fig 3.6: Login Screen

3.3.6.3 Application Description

This Application is built on the Microsoft Windows 2007 platform using Java web system and XAMMP. The application is designed in the form of an interface (client side) ï server model. Server being used is glassfish server 3.2, MySQL server for database. The server side is built using java 7 and jsf(2.0). The client side is built on html5, css3, jQuery 1.7, JavaScript and prime faces. In the access control class, the clients play the role of users and the server that of the key and data server, supplying both keys and allowing access to data. In the application, the server generates AES (Advanced Encryption Standard) encryption/decryption keys and keeps an encrypted log file of the encrypted text files. Logs file were also implemented on the application. For logs xml files and a java api from the log4j library was used. Logs indicate a date and time, add the class that we are in, indicate what is being logged i.e. error, warning or info and indicate a message to further elaborate what is happening on the application.

In this research, a file uploading has being used as service to the web application. The security is applied over the data at the background using the encryption algorithm AES.

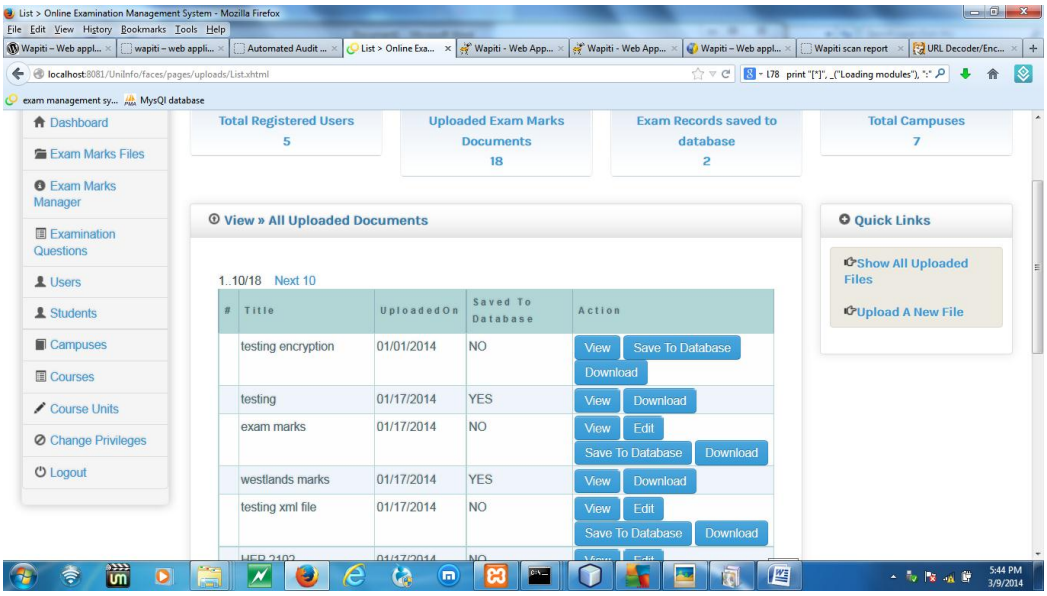


Fig 3.7: Uploading a Document

3.3.6.4 Uploading a document

- Website page is local host: 8081/UniInfo/
- Go to the login in page and enter the username and password.
- Here one can access the Dashboard on the main page showing ï Total Registered users, Uploaded Exam Marks Document, Exam records saved to database, Total Campuses, Upload files and user privileges.
- Download a Microsoft excel template where exam results can be saved and save it on a folder. In this case the template is saved at a folder called systems on the desktop.
- Go to the Exam Marks Manager and on the Quick Links, click on upload a New File.
- On the upload File, enter the unit name of the exam results being upload and the course description.
- Click on the browse tab and locate the template of the exam results from the system folder and upload it.
- Confirm if the file is uploaded on the uploads folder: UniInfo ï Web ï Resources ï Uploads.
- This show the time and date of when the file was uploaded.
- At this point the file is encrypted using secret key and if someone tries to access the file a message appears on the screen saying the file is corrupted. When opened the file appears as a cipher text and the character cannot be read.

- Go to xml document and open the file written security. Here the uploaded file is shown with the secret key used to encrypt the file and the time and date. The application generated its own secret keys that are used to encrypt or decrypt the data. No one can have access to the keys. One cannot access the document once itôs uploaded and hence security of the uploaded document is guaranteed.
- To download an upload file, the application generated a secret key, which is used to decrypt the file into a plaintext.
- Only registered users can access this web page. The details are saved in the database server and every time one logs on the page, it can verify his/her credentials.

Uploaded document on the system is shown in figure 3.8 below:

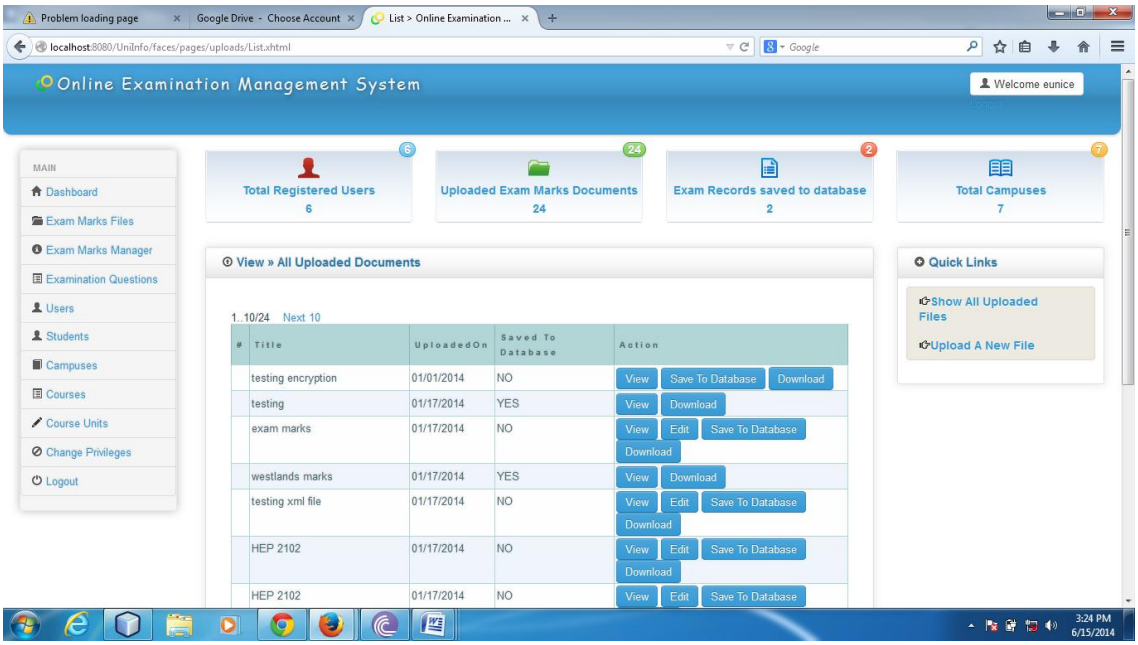


Fig 3.8: Uploaded Documents

After processing, the encrypted document appears as shown in figure 3.9 below:

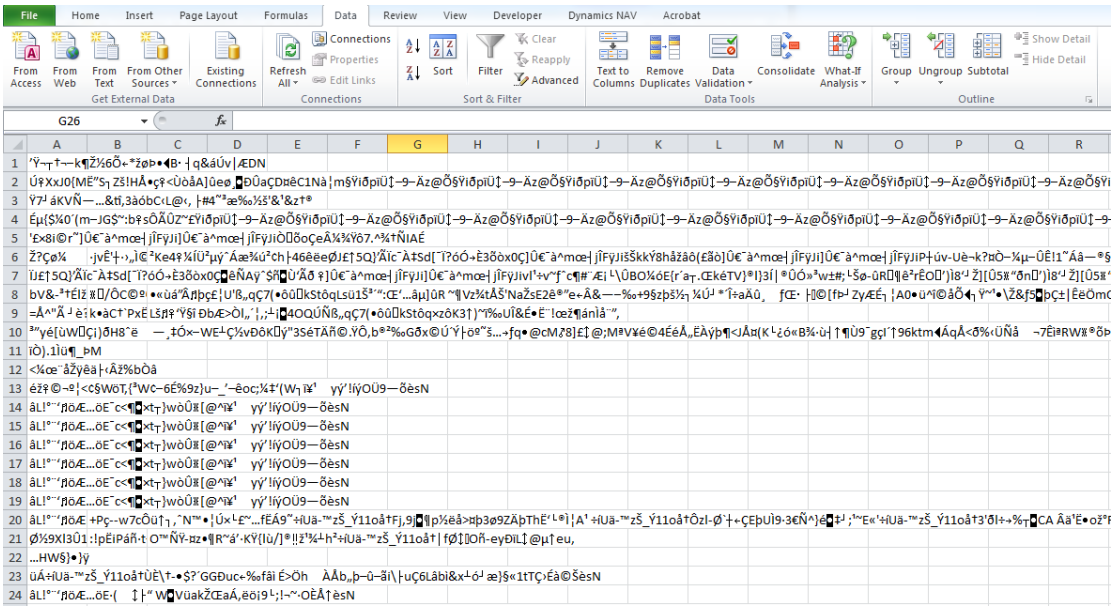


Fig 3.9: Encrypted Document

CHAPTER FOUR
EVALUATION

4.1 Overview

This phase deals with the actualization of the prototype system. The prototype system is implemented as a web based database driven system. The codes are presented in Appendix 2. This section highlights how the system was tested. System testing is aimed at evaluating the system to determine if it meets the required results. Examination text files were uploaded on the prototype using symmetric keys, the plaintext file was encrypted and the cipher text file was stored in the MySQL database. When the text files were downloaded, the symmetric key was used to decrypt the cipher text to plaintext. Inbuilt java security codes were used to ensure that the symmetric keys were safely stored and no unauthorized person could get access to the keys.

4.2 Evaluate performance

4.2.1 Experiments Done

The main aim of the experiment was to encrypt and upload text files and then confirm vulnerability of the encrypted files. Examination text files were uploaded on the prototype using dynamic symmetric keys, the plaintext file was encrypted and the cipher text file was stored in the MySQL database. A statistical formula was used to determine the number of logs that are required to analyze the prototype system. The number of students taking computer science was taken as the population of the pool. From the pool a sample of logs was taken. Assumptions were made that in the 10 satellite campuses in JKUAT each campus has approximately 100 students taking computer related courses. The computer course takes roughly 4 years to complete so 4 streams were used and that gave the population of 4000 students in the campuses. These students were assumed to have the basic technical knowledge of how to try and access the system without authorization. Using 142 logging attempts at the sample size that were used to access the prototype, the research was able to analyze the prototype.

Text files downloaded used the symmetric key to decrypt the cipher text to plaintext. Because of the dynamic nature of the key generator, the experiments show that it is difficult to crack the key in order to access the encrypted data. The logs were analyzed before the key generator was implemented in the prototype and after implementing the key generator. Fig

3.3 show the Number of login failures after the key generator was implemented was 105, operation problems was 17 and potentially dangerous files was 17. This is out of the 142 logs taken as the sample size. Figure 3.4 shows the number of unauthorized access users in the different campuses before the key generator was implemented and after it was implemented in the prototype. This analysis clearly shows that after the key generator was implemented, the number of users trying to access the prototype was reduced tremendously.

Figure 3.5 shows the number of users who managed to access the prototype and tried to decrypt the uploaded text files. This shows that after implementing the key generator in the prototype, the number has also reduced. This implies that as much as the users were able to access the prototype, it was not easy to access the encrypted text files hence the prototype is very secure. Further analysis were done on the logs after the key generator was implemented and 48% showed the entered login page, 40% showed failed decryption attempt and 12% showed redirecting users according to the pie chart in figure 3.6 . This shows that the prototype is not vulnerable to unauthorized users. This experiments show that AES based model with key generator has improved the security aspect of the prototype and therefore text files can be uploaded in an encrypted format to ensure unauthorized users donôt get access to the files.

4.2.2 Sampling Method

Bayesian statistics was used to determine the number of logs that are required to analyze the prototype system. The number of students taking computer science was taken as the population of the pool. From the pool a sample of logs was taken. Assumptions were made that in the 10 satellite campuses in JKUAT each campus has approximately 100 students taking computer related courses. The computer course takes roughly 4 years to complete so 4 streams were used and that gave the population of 4000 students in the campuses. These students were assumed to have the basic technical knowledge of how to try and access the system without authorization.

Table 4.1 ï Showing the Sample Size used to analyze the Logs

Nos	**Satellite Campus**	**Nos of Students**
1	Main Campus	400
2	Westlands Campus	400
3	Mombasa Campus	400
4	Nairobi Campus	400
5	Nakuru Campus	400
6	Kisii Campus	400
7	Voi Campus	400
8	Karen Campus	400
9	Kisumu Campus	400
10	Arusha Campus	400
TOTAL POPULATION		**4000**

Using Bayesian statistics, a sample size was gotten as per the formula below.

The following formula will be applied to determine the sample size.

$$n = \frac{z^2. .pq.N}{e^2. (N-1)+z^2 .p.q}$$

Where:

N = Size of the population

n = Sample size

e = Margin of error / precision

z = Standard variant at a given confidence level

p = the proportion of the population estimated to have characteristics being measured

q = 1-p

In the research, the population of Computer science students was used as a representative of the all the malicious attackers who might have an interest to access the examination system.

In-order to determine the standard variant, usually precision refers to 95% confidence level of the true value that particular effect. By using tables of normal probabilities, the z value for the 95% confidence level is 1.96.

The precision of the research is estimated to be + or ï 5% and therefore;

z = 1.96, e=0.05

Hence

P = 0.09524

N= 4000

q =1-p=(1-0.09524)= 0.90476

z=1.96, e-=0.05

Therefore estimated sample size is

$$\frac{1.96^2 \text{ x } 0.09524 \text{ x } 0.90476 \text{ x } 4000}{0.05^2 \text{ x } (4000\text{-}1) + 1.96^2 \text{ x } 0.09524 \text{ x } 0.90476}$$

142.2 = approximately 142 students

Bayesian statistics is a good method for getting a sample from a large population.

Using 142 logging attempts that were used to access the prototype, the research was able to analyze the prototype.

4.3 Results and Analysis

The logs collected were analyzed, vulnerabilities were analyzed and the report produced was stated as shown in the graph below. A statistical formula was used to get a sample size of 142 logs which was used to analyze the prototype. Fig 3.3 shows the log analysis of the 142 sample size after the key generator was implemented. The figure show 105 login failures, 20 operation problems and 17 potentially dangerous files. Out of the 142 users who randomly accessed the prototype, 105 did not manage to login into the prototype due to lack of the appropriate credentials. 20 managed and did not attempt to access the uploaded documents. The remaining 17 managed to access the prototype and attempted to access the encrypted documents but did not manage. See fig 3.3 below.

Table 4.2 ï Showing the Login Failures, Operational Problems and Potentially Files.

Login Failures	Operational problems	Potentially dangerous files	Total Users
105	20	17	142

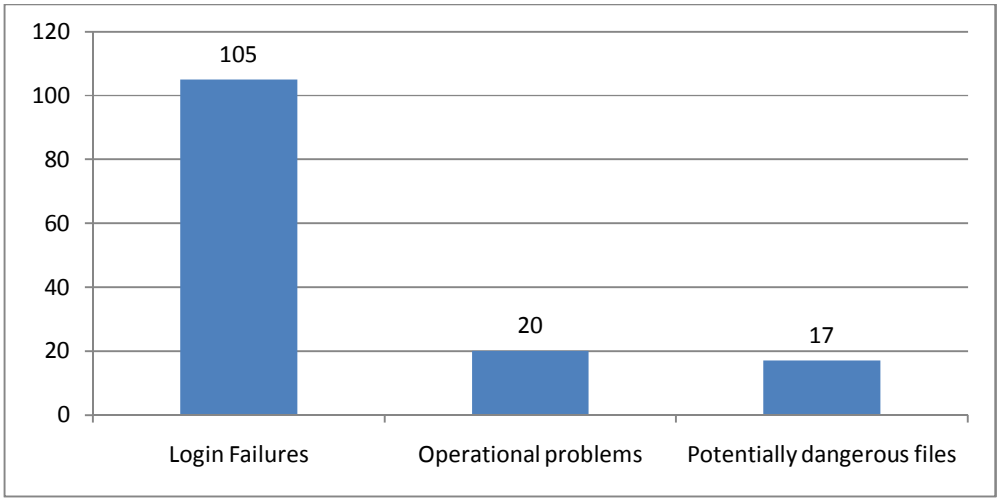


Fig 4.1: Login analysis of 142 users

4.3.1 Analysis of the number of unauthorized users

The logs were analyzed before the key generator was implemented and after the key generator was implemented in the prototype. Fig 4.2 shows the campuses and the number of unauthorized access before and after the key generator was implemented. From the figure the number has reduced compared to when the key generator was not implemented.

Table 4.3 ï Showing attempts based of different Satellite Campus.

Nos	Satellite Campus	Nos of Students	passive (unauthorized access)	active attacks
1	Main Campus	39	39	
2	Westlands Campus	24		24
3	Mombasa Campus	11	11	
4	Nairobi Campus	19		19
5	Nakuru Campus	8		8
6	Kisii Campus	7		7
7	Voi Campus	8	8	
8	Karen Campus	9	9	
9	Kisumu Campus	13		13
10	Arusha Campus	4		4
TOTAL POPULATION		**142**	**67**	**75**

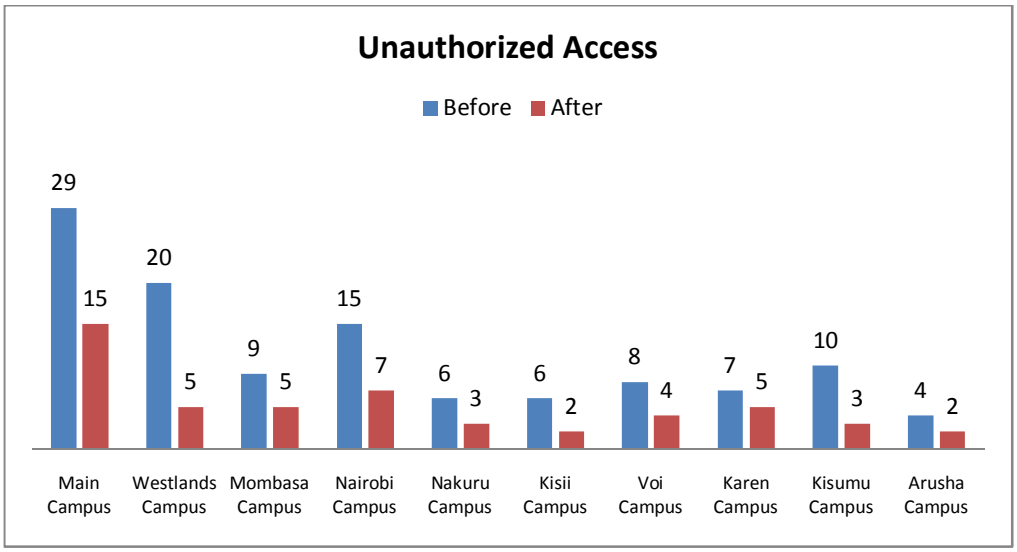


Fig 4.2: Analysis of the number of unauthorized users

4.3.2 Analysis of the number of Decryption attempt users

Out of the 142 users who randomly accessed the prototype, Fig 4.3 shows the number of users who tried to decrypt the uploaded text files before the key generator was implemented

and after it was implemented. The figure show that the decryption attempts by users reduced after the improved AES based model with key generator.

Table 4.4 ï Showing attempts on access on encrypted Documents before and after Encryption.

Satellite Campus	Before	After	Before	After
Main Campus	29	15	15	5
Westlandôs Campus	20	5	14	6
Mombasa Campus	9	5	8	3
Nairobi Campus	15	7	15	5
Nakuru Campus	6	3	6	2
Kisii Campus	6	2	5	1
Voi Campus	8	4	5	1
Karen Campus	7	5	8	2
Kisumu Campus	10	3	9	4
Arusha Campus	4	2	3	1
	114	**51**	**88**	**30**

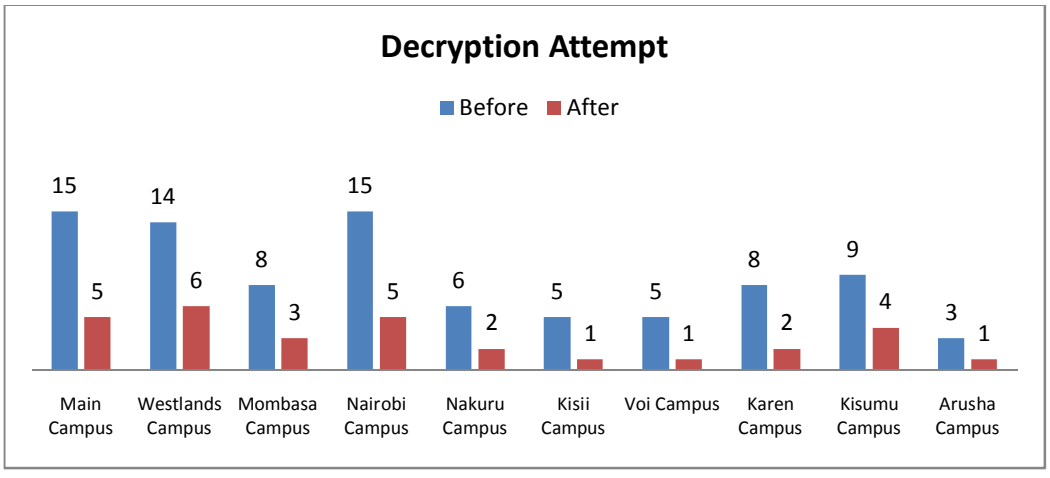


Fig 4.3: Analysis of the number of Decryption attempt users

4.3.3. Vulnerability report of the logs

An analysis of the overall login sessions in the prototype was done to determine the vulnerability of the prototype. Fig 4.4 shows the percentages of entered login page, failed

decryption attempt and redirecting user. This generally shows that even if the unauthorized users managed to login in the prototype there were not able to access the encrypted text files and that was the main aim of the experiments.

Table 4.5 ï Vulnerability report of the logs

Entered Login page	Failed decryption attempt.	Redirecting user	Total Users
20	17	5	42

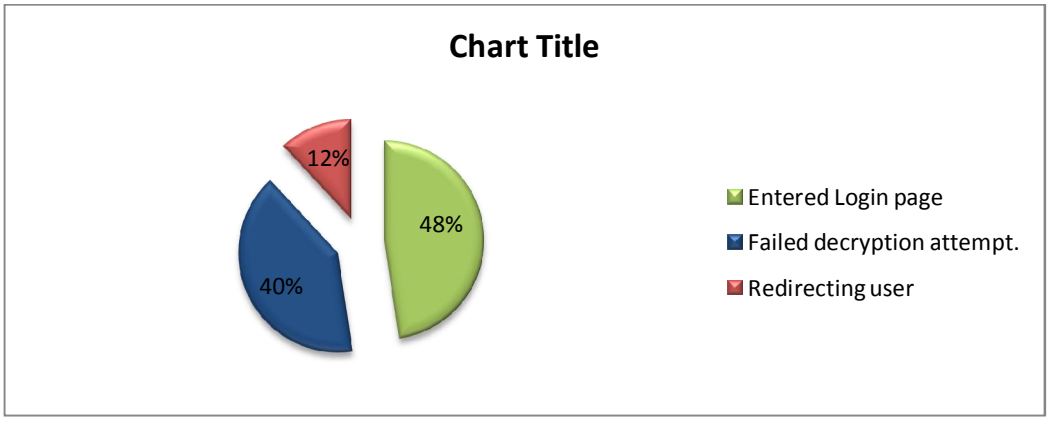


Fig 4.4: Vulnerability report of the logs

4.4 Summary

Fig 4.1 shows the Number of login failures after the implementation of the key generator to be 105, operation problems 20 and potentially dangerous files 17. This is out of the 142 logs taken as the sample size. Figure 4.2 shows the number of unauthorized access users in the different campuses before the implementation and after implementation of the prototype. This analysis clearly shows that after the key generator was implemented, the number of users trying to access the prototype reduced tremendously. Figure 4.3 shows the number of users who managed to access the prototype and tried to decrypt the uploaded text files. This shows that after implementing the key generator in the prototype, the number reduced. This implies that as much as the users were able to access the prototype, it was not easy to access the encrypted text files hence the prototype is very secure. Further analysis were done on the logs after the key generator was implemented and 48% showed the entered login page, 40%

showed failed decryption attempt and 12% showed redirecting users according to the pie chart in figure 4.4 .

CHAPTER FIVE
CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

In this digital world, which is currently evolving and changing at such a rapid pace, the security of digital information has become increasingly more important. Encryption is important for the protection of secret files and document from unauthorized access. Having said that, there is need for a secure model that facilities secure transmission of examination data from one person to another. In transmission of examination data over the Internet, the data is transferred between the server and client. Data is secured in the transmission channel using a strong encryption algorithm and server where it is stored, based on usersô choice of security method. This model will prevent unauthorized access to server, which stores examination data within the examination department in the Universities. With the increase of geographically distributed campuses within Universities, there is need to secure the data that is transferred from one campus to another. However there are limited measures when it comes to securing this data. Examination data is very sensitive and access to it by unauthorized personnel might result to stakeholders losing confidence in the examination process. This has necessitated the need for a secure way to transmit this data from one computer to another computer.

The research looked at the possibility of using an improved encryption algorithm developed based on private key encryption, which provides a secure communication between two computers. The objectives of the research were to analyze the current Examination System Management and the existing encryption algorithms used to secure data in transit. The research also aimed at developing an encryption model, which provides an extension called a guard, which improves AES encryption algorithm. The algorithm uses key generation method in algorithm for increasing efficiency of algorithm. A prototype was then developed based on the model. The prototype was to be the basis of testing the model. The final objective was to test the model and evaluate its performance. The developed prototype allowed users to upload encrypted text document using dynamically generated keys and decrypt the downloaded text documents. 10 campuses used the prototype and the results were analyzed based on that.

Cryptography algorithms are classified in two types, public-key producing and symmetric-key producing algorithms. In this research, an improved encryption algorithm was developed based on private key encryption, which provides a secure communication between two users with the developed algorithm. The encryption algorithm method that was found suitable to use for encryption of data was Advanced Encryption algorithm. This encryption algorithm uses enhanced symmetric key in which same structure of encryption and decryption is used. The model provides an extension called a guard, which improves AES encryption algorithm. The algorithm use key size of 128 bits for providing better security and it also provide the concept of internal key generation at receiver end on the basis of 128 bits key which will be the receiver end by other path for preventing harmful attacks on the model. One challenge with AES encryption is the exchange of the secret key between the participating parties. Since both encryption and decryption use the same secret key, an attacker might get the key and comprise the data being transmitter. This research developed a guard that generates dynamic keys so as to improve the security of the AES.

Using a statistical method a sample size was determined from 10 campuses who were allowed to test the prototype. An online examination management prototype model was developed. A simple interface was designed and developed using java web system to provide an easy and fast user experience for computer users. An encryption algorithm was used to encrypt plaintext into cipher text for secure transmission of examination text files. The user was able to upload text files on the prototype and using secret keys the text files are encrypted and stored in a MySQL database. The prototype then uses the secret key and decrypts the cipher text into plaintext in the case of decryption. The prototype was divided into two phases i.e. enrollment phase and login phase. At the enrollment phase a new user create an account, which he uses for logging in the system. At the login phase the user inputs his username and password to enable him or her upload or download a document.

The prototype was designed to have an inbuilt log analyzer to access who or what was being accessed in the prototype. Using the log analyzer, the application can automate the entire process of managing generated logs by collecting, analyzing, searching, reporting and archiving from one central location. This event log analyzer software helps to mitigate internal threats, monitor file integrity, monitor privileged users and intelligently analyze the logs to generate a variety of reports like user activity reports, regulatory compliance reports and historical trend reports. A statistical method was used to determine the sample size of the

security logs collected by the prototype. Because of the dynamic nature of the key generator, the experiments show that it is difficult to crack the key in order to access the encrypted data. The logs were analyzed before the key generator was implemented in the prototype and after implementing the key generator. The analysis shows the Number of login failures after the key generator was implemented was 105, operation problems were 17 and potentially dangerous files were 17. This is out of the 142 logs taken as the sample size. The number of unauthorized access users in the different campuses before the key generator was implemented and after it was implemented in the prototype was analyzed. The analysis clearly shows that after the key generator was implemented, the number of users trying to access the prototype was reduced tremendously. Further experiments showed that the number of users who managed to access the prototype and tried to decrypt the uploaded text files had also reduced. Further analysis were done on the logs after the key generator was implemented and 48% showed the entered login page, 40% showed failed decryption attempt and 12% showed redirecting users.. This shows that the prototype is not vulnerable to unauthorized users. These experiments show that AES based model with key generator has improved the security aspect of the prototype and therefore text files can be uploaded in an encrypted format to ensure unauthorized users donôt get access to the files.

The research was therefore able to achieve its objectives of developing a model that could secure examination data by encrypting and decrypting transmitted data. The model also confirmed that it was possible to encrypt examination documents and save them in an encrypted format to avoid unauthorized access of the document. The evaluation confirmed that all the attempted access to the model were captured and therefore the model could monitor who had access to the prototype and at what time. This means that the guard in the AES could be relied upon and an encryption algorithm to upload examination documents in a safe way.

The findings of this research project will be of great importance to the Universities as it would come up with a secure system for ensuring that examination data is transferred in a secure manner free of comprising the examinations materials. This ensures that examinations are handled in a secure manner putting in mind standards such as confidentiality, integrity and availability. This model will prevent unauthorized entries to systems, which stores

examination data within the examination department in the Universities hence improving the confidence in this department.

The major contributions of this research are: The developed prototype allows users to encrypt and send their text files or decrypt and read the received ones in a secure manner using Encryption. Due to the increasing numbers of distributed satellite campuses there is a need to transmit examination data from one computer to another via the Internet. To ensure the integrity of the examination data transmitted is maintained between the sender and the receiver, the encryption algorithm with secure keys will be implemented for secure data transmission. We can also extend the application scope of this algorithm to other departments e.g. finance, health information etc.

5.2 Recommendations for Further Research

There is need to formulate a model that is generic enough to ensure secure examination data transmission over the Internet. This project was able to come with a guard that improves the security on the existing AES encryption algorithm. Finally, further research is needed to come up with improved encryption algorithm in order to ensure secure transmission of data is ensured at all times. Also research on Public key encryption and security of examination data is also needed as public key encryption is more secure than private key encryption though it requires more computation and system resources.

REFERENCES

Abdul Wahid Soomro, N. I. (2010). Secured Symmetric Key Cryptographic Algorithmfor Small

Amount of Data. *International Conference on computer & emerging technology*.

Douglas H (2009) Standards for secure data sharing across organizations. The University of Texas at Dallas, United States The Department of Defense, U.S.

Camp, L.J (2003). "Identity in digital government ", a research report of the Digital Government .

Civic Scenario Workshop, Kennedy School of Government, Harvard University, Cambridge.

Crompton, M .(2004). "Proof of ID required? Getting identity management right", paper presented at Australian IT Security Forum, Sydney, 30 March.

Allam Mouse & Ahmad Hamad (2006). Evaluation of the RC4 Algorithm for Data Encryption. Electrical Engineering Department, An-Najah University, Nablus Palestine.

Ankara. (2007). *Comparing Two Prominent Variants of RSA Cryptography System.* india.

Ayushi. (2010). A Symmetric Key Crytographic algorithm. *International journal of computer applications*.

Biham. (1993). New types of crytanalytic attacks using related keys, Advances in Cryptology. *Springer-Verlag*, 398 -409.

Bruce Schneier, J. K. (1999). Performance Comparison of the AES Submissions. *NIST*, 17.

Commerce, D. o. (2007). security guide for information systems.

Agrawal, R . (2003). Information Sharing across Private Databases. In: 10th International Conference on Management of Data, pp 86-97.

Dan Boneh, A. S. (2006). Functional Encryption: Definations and Challenges.

Monika Agrawal, Pradeep Mishra. (2012). A Modified Approach for Symmetric Key Cryptography Based on Blowfish Algorithm. International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-1, Issue-6.

Elbaz, L. (2002). Using Public key Crytography in Mobile Phones. *Discretix Technology Limited*.

Eric Brier, a. e. (2006). Correlation power analysis with data leakages.

Gary Locke, P. G. (2009). Digital Signature Standard. *Federal Information processing Standards Publication*.

Bilham & Shamir .(1993). A Differential Cryptanalysis of the Data Encryption Standard, Springer Berlin Heildelberg New York, 1993.

Dinesha H.A & Prof Agrawal. (2013). Framework design of secure cloud Transmission protocol.

Hangzhou, Zhejiang. (2013). Algorithm design of secure data message transmission based on openSSL and VPN. Department of Computer and Information Engineering, Zhejiang Water Conservancy and Hydropower College.

Hyubgun Lee, K. Y. (2009). AES Implementation and performance.

Jerome Burke, J. M. (2009). Architectural support for Fast Symmetric-Key Cryptography. *Advance Computer Architecture Laboratory*.

Jonathan, B. (2000). A Chosen Ciphertext Attack Against Several E-Mail Encryption .

Krishna Kumar Pandey, V. R. (2013). An Enhanced Symmetric Key Crytography Algorithm to Improve Data Security. *International Journal of Computer Applications*.

krishna kumar, v. R. (2013). An Enhanced symmetric key crytography algorithm to improve data security. *International Journal of computer Applications*.

Jonathan Katz & Bruce Schneier. (2000). A Chosen Ciphertext Attack Against Several Email Encryption protocols.

Mahalanobis, A. (2005). Diffie-Hellman Key Exchange Protocol.

Nursel Yalcin & Utkukose. (2006). Sending E-Mail with an encrypting algorithm based on private -key encryption.

Menezes, A., & Van Oorschot, P. (1996). *Handbook on Applied Crytography.*

Monika. A, P. M. (2012). A modified approach for symmetric key Crytography Based on Blowfish Algorithm,. *IJEAT*.

Oded, G. (2004). Foundations of Crytography Vol 2. *Cambridge University Press*.

Palmgren, K. (2006). Diffie-Hellman Exchange - Non Mathematicians Explaination.

National Institute of standards and technology(U.S), Advanced Encryption Standard (AES).

Prakash, G. D. (2013). A Symmetric Key Algorithm for secured Information Exchange Module. *Computer Network and Information Security*, 37-45.

Ramaratan Rathore, C. (2014). Verification of data intergrity. *Institute of Engineering and Technology*.

Randall, N. &. (1999). Defending your digital assets against hackers, crackers, spies and thieves.

Rijmen, J. D. (2002). *The design of Rijndael, AES - The Advanced Encryption Standard.* Springer-Verlag.

Saranya, M. (2014). A Review on Symmetric Key Encryption Techniques in Crytography. *International Journal of Science, Engineering and Technology Research (IJSETR)*, Volume 3.

Sayed Tathir Abbas, K. (2013). Analytical Study of AES .

Sonipat, H. (2010). A Symmetric Key Cryptographic. *International Journal of Computer Application*, Volume 1 – No. 15.

Srikumar V & others(2008) A Taxonomy of Data Grids for Distributed Data Sharing, Management and processing Grid Computing and Distributed Systesms Laboratory

Stallings, W. (2003). Cryptography and network security. *apprentice hall*.

Stephen F. Bisbee, D. H. (1997). System and method for eletronic Transmission strorage & retrieval of authenticated documents.

Suyash Verma, e. a. (2012). An efficient developed new Symmetric Key Crytography Algorithm for Information Security. *IJETAE*.

Wang Bo al, e. (2011). information security technology in the current market. *IEEE*.

Yin, X. (2003). Issues of security protocols in relation to encryption.

William Stallings. (2003). Cryptography and network security: Principles and practice, apprentice hall, upper saddle river, New Jersey.

APPENDICES

Appendix 1- Source Code to generate Security Logs

Shown below is the source code for generating security logs of the java based prototype:

```
</ Code to generate security codes/>
<? xml version="1.0" encoding="UTF-8" ?>
<! DOCTYPE log4j: configuration SYSTEM "log4j.dtd">
<log4j: configuration xmlns: log4j='http://jakarta.apache.org/log4j/'>
<appender name="CA" class="org.apache.log4j.ConsoleAppender">
<layout class="org.apache.log4j.PatternLayout">
<param name="Conversion Pattern" value="%-4r [%t] %-5p %c %x - %m%n" />
</layout>
</appender>
<appender name="FA" class="org.apache.log4j.FileAppender">
<param name="File" value="/C:/log Files/UniInfo.log"/>
<param name="Threshold" value="INFO"/>
<layout class="org.apache.log4j.PatternLayout">
<param name="ConversionPattern" value="%d{dd MMM yyyy HH:mm:ss,SSS} [%t] %-5p %c      %x - %m%n" />
</layout>
</appender>
<root>
<level value="DEBUG" />
<appender-ref ref="CA" />
<appender-ref ref="FA" />
</root>
</log4j: configuration>
```

Appendix 2- Source code for the guard

Shown below is the source code for the guard on java based prototype:

```
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package com.JSFManagedBeans.kigaita;

import com.JSFClasses.kigaita.util.FileEncryptionClass;
import com.JSFClasses.kigaita.util.LogClass;
import com.entityClasses.kigaita.Uploads;
import com.JSFSessionBeans.kigaita.UploadsFacade;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.Serializable;
import java.net.MalformedURLException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.sql.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.BadPaddingException;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.NoSuchPaddingException;
import javax.ejb.EJB;
import javax.faces.application.FacesMessage;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.bean.ViewScoped;

import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import login.LoginManagedBean;
import org.apache.commons.io.FilenameUtils;
import org.apache.log4j.xml.DOMConfigurator;
import org.primefaces.model.UploadedFile;

/**
 *
 * @author kigaita
 */
@ManagedBean
@RequestScoped
public class UploadsManagedBean implements Serializable {

    @EJB
private UploadsFacade uploadsFacade;
```

```
private UploadedFile fileName = null;
private String title, Description, nameOfFile;
public int uploadIdParam;
   //private int id;
   FacesContext fc = FacesContext.getCurrentInstance();
private int uploadedBy, id;
private int savedToDB;
private Date uploadedOn;
private String addNameToFileName;
private FileEncryptionClass fileEncryptionClass ;

static final org.apache.log4j.Logger logger = org.apache.log4j.Logger.getLogger(UploadsManagedBean.class);

public UploadsManagedBean() {
DOMConfigurator.configure(LogClass.getLogFilePath());
}

public int getUploadIdParam() {
return uploadIdParam;
   }

public void setUploadIdParam(int uploadIdParam) {
      this.uploadIdParam = uploadIdParam;
   }

public String getNameOfFile() {
return nameOfFile;
   }

public void setNameOfFile(String nameOfFile) {
      this.nameOfFile = nameOfFile;
   }

public static String getUrlBase() {
      StringBuilder path = new
StringBuilder(FacesContext.getCurrentInstance().getExternalContext().getRealPath("/resources//"));
int indexOfBuildString = path.indexOf("build\\");
int lengthOfBuildString = "build\\".length();
path.delete(indexOfBuildString, indexOfBuildString + lengthOfBuildString);
     //String
path=FacesContext.getCurrentInstance().getExternalContext().getRequestContextPath()+File.separator+"resou
rces";

return path.toString();
   }
private String destination = getUrlBase();

public void copyFile(String fileName, InputStream in) {
try {
         // write the inputStream to a FileOutputStream
         OutputStream out = new FileOutputStream(new File(destination + "//uploads//" + fileName));
int read = 0;
byte[] bytes = new byte[1024];
while ((read = in.read(bytes)) != -1) {
out.write(bytes, 0, read);
         }
in.close();
```

```
out.flush();
out.close();
        File renamingFile = new File(destination + "//uploads//" + fileName);
if (renamingFile.exists()) {
System.out.println("################## FILE EXISTS /n " + renamingFile);
        } else {
System.out.println("################## FILE DOES NOT EXISTS");
        }

System.out.println("################## FILE DOES NOT EXISTS TESTED URL" + destination + "//uploads//" +
fileName);
System.err.println("Rename returned " + this.getAddNameToFileName().toString());
renamingFile.renameTo(new File(destination + "//uploads//" + this.getAddNameToFileName() + fileName));
System.out.println("New file created!");
logger.info("File "+fileName+" uploaded to the server successfully");
      } catch (IOException e) {
logger.error("Could not upload  "+fileName+" to server. ");
logger.error(e.getMessage());
System.out.println(e.getMessage());
      }
   }

public void copyFile() {
      File fileToEncrypt =null;
      File encryptedFile=null;
try {

        OutputStream out = new FileOutputStream(new File(destination + "//uploads//" +
fileName.getFileName()));
int read = 0;
        InputStream in=fileName.getInputstream();
byte[] bytes = new byte[1024];
while ((read = in.read(bytes)) != -1) {
out.write(bytes, 0, read);
        }
in.close();
out.flush();
out.close();

fileToEncrypt=new File(destination + "//uploads//" + fileName.getFileName());

encryptedFile = new File(destination + "//uploads//" + getAddNameToFileName() + fileName.getFileName());
fileEncryptionClass = new FileEncryptionClass();

fileEncryptionClass.encryptyData(fileToEncrypt, encryptedFile);
logger.info("File "+fileName+" uploaded to the server and encrypted successfully");

      } catch (NoSuchAlgorithmException ex) {
Logger.getLogger(UploadsManagedBean.class.getName()).log(Level.SEVERE, null, ex);
logger.error("Error encountered while uploading "+fileName+" to the server.");
logger.error(ex);

      } catch (NoSuchPaddingException ex) {
Logger.getLogger(UploadsManagedBean.class.getName()).log(Level.SEVERE, null, ex);
logger.error("Error encountered while uploading "+fileName+" to the server.");
logger.error(ex);
      } catch (InvalidKeyException ex) {
```

```
Logger.getLogger(UploadsManagedBean.class.getName()).log(Level.SEVERE, null, ex);
logger.error("Error encountered while uploading "+fileName+" to the server.");
logger.error(ex);
      } catch (IllegalBlockSizeException ex) {
Logger.getLogger(UploadsManagedBean.class.getName()).log(Level.SEVERE, null, ex);
logger.error("Error encountered while uploading "+fileName+" to the server.");
logger.error(ex);
      } catch (BadPaddingException ex) {
Logger.getLogger(UploadsManagedBean.class.getName()).log(Level.SEVERE, null, ex);
logger.error("Error encountered while uploading "+fileName+" to the server.");
logger.error(ex);
      } catch (IOException ex) {
Logger.getLogger(UploadsManagedBean.class.getName()).log(Level.SEVERE, null, ex);
logger.error("Error encountered while uploading "+fileName+" to the server.");
logger.error(ex);
      }
finally{
if(fileToEncrypt.exists())
fileToEncrypt.delete();

      }
   }

public void uploadingFile() {
      Random rand = new Random();
      DateFormat dateFormat = new SimpleDateFormat("yyyymmdd");
setAddNameToFileName(dateFormat.format(this.getUploadedOn()) + "_" + this.getUploadedBy() + "_" +
rand.nextInt(Integer.MAX_VALUE) + 1 + "_");
      ExternalContext externalContext = FacesContext.getCurrentInstance().getExternalContext();
      String ext = FilenameUtils.getExtension(fileName.getFileName());
if (fileName != null && (ext.equalsIgnoreCase("xls") || ext.equalsIgnoreCase("xlsx"))) {

         //copyFile(fileName.getFileName(), fileName.getInputstream());
copyFile();
System.err.println("File " + fileName.getFileName() + " is uploaded.");
         FacesMessage msg = new FacesMessage("Succesful '\n", fileName.getFileName() + " has been
uploaded.");
FacesContext.getCurrentInstance().addMessage(null, msg);


      } else if (!(ext.equalsIgnoreCase("xls") || ext.equalsIgnoreCase("xlsx"))) {
fileName = null;
System.err.println("File  Format uploaded is not supported.");
         FacesMessage error = new FacesMessage("File uploaded is not a valid excel file!");
FacesContext.getCurrentInstance().addMessage(null, error);
logger.error("File uploaded is not a valid excel file!");
      } else {
fileName = null;
System.err.println("File   Not  uploaded.");
logger.error("User did not provide a file to be uploaded. ");
         FacesMessage error = new FacesMessage("The files were not uploaded!");
FacesContext.getCurrentInstance().addMessage(null, error);
      }
   }

public void createUpload() {
this.uploadingFile();
```

```
if (fileName != null) {
        FacesMessage msg = new FacesMessage("Succesful '\n", fileName.getFileName() + " has been
uploaded.");
FacesContext.getCurrentInstance().addMessage(null, msg);
        Uploads uploadEntity = new Uploads(null, this.getTitle(), getAddNameToFileName() +
fileName.getFileName(), this.getUploadedBy(), this.getUploadedOn(), 0, this.getDescription());
uploadsFacade.create(uploadEntity);
      }
   }

public String getAddNameToFileName() {
return addNameToFileName;
   }

public void setAddNameToFileName(String addNameToFileName) {
      this.addNameToFileName = addNameToFileName;
   }

public UploadsFacade getUploadsFacade() {
return uploadsFacade;
   }

public void setUploadsFacade(UploadsFacade uploadsFacade) {
      this.uploadsFacade = uploadsFacade;
   }

public int getSavedToDB() {
return savedToDB;
   }

public boolean getSavedToDBStatus(int saved) {
if (saved == 1) {
return true;
      } else {
return false;
      }
   }

public void setSavedToDB(int savedToDB) {
      this.savedToDB = savedToDB;
   }

public UploadedFile getFileName() {
return fileName;
   }

public void setFileName(UploadedFile fileName) {
      this.fileName = fileName;
   }

public String getTitle() {
return title;
   }

public void setTitle(String title) {
      this.title = title;
   }
```

```
public String getDescription() {
return Description;
   }

public void setDescription(String Description) {
      this.Description = Description;
   }

public int getUploadedBy() {
      FacesContext context = FacesContext.getCurrentInstance();
      //loginBean= context.getExternalContext().getSessionMap().get("userId");
FacesContext.getCurrentInstance().getExternalContext().getSessionMap().get("userId");
return context.getExternalContext().getSessionMap().get("userId").hashCode();
   }

public void setUploadedBy(int uploadedBy) {
      this.uploadedBy = uploadedBy;
   }

public java.util.Date getUploadedOn() {

return new java.util.Date();
   }

public void setUploadedOn(Date uploadedOn) {
      this.uploadedOn = uploadedOn;
   }

public int getId() {
return id;
   }

public void setId(int id) {
      this.id = id;
   }

public String getDestination() {
return destination;
   }

public void setDestination(String destination) {
      this.destination = destination;
   }

public List<Uploads> getAllUploads() {

return uploadsFacade.findAll();
   }

public void ViewUpload() {
      String url;
FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap().get("uploadId");
System.err.println("Your Id " +
FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap().get("uploadId"));
```

```
if
(Integer.valueOf(FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap().get("upl
oadId")) != null) {
logger.info("redirecting user to view uploads page");
url = "./uploads/ViewUpload.xhtml";
      } else {
logger.info(" No file selected. Redirecting user to profile page");
url = "./profile.xhtml";
      }

//this.getUpload(Integer.parseInt(FacesContext.getCurrentInstance().getExternalContext().getRequestParamet
erMap().get("uploadId")));

doRedirect(url);
   }

public void EditUpload() {
FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap().get("uploadId");
System.err.println("Your Id " +
FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap().get("uploadId"));
      //this.getUpload();

//this.getUpload(Integer.parseInt(FacesContext.getCurrentInstance().getExternalContext().getRequestParamet
erMap().get("uploadId")));
      String url = "../uploads/EditUpload.xhtml";
logger.info("Redirecting user to edit uploads page.");
doRedirect(url);
   }

public int countUploads() {

return uploadsFacade.count();
   }

public void doRedirect(String url) {
try {
FacesContext.getCurrentInstance().getExternalContext().redirect(url);
      } catch (IOException ex) {
ex.printStackTrace();
      }
   }
}
```

Appendix 3- Source Code for the Dynamic Generation of secret keys.
Shown below is the source code for the dynamic generation of secret keys based on java prototype:

```
* To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package com.JSFClasses.kigaita.util;

import com.JSFManagedBeans.kigaita.FileDownloadController;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.math.BigInteger;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.apache.log4j.xml.DOMConfigurator;
import org.omg.PortableInterceptor.SYSTEM_EXCEPTION;

/**
 *
 * @author Kigaita
 *
 *
 * used for encryption
 */
public class FileEncryptionClass extends XmlUtil {

private CipherOutputStream outputStreamCipher;
private CipherInputStream inputStreamCipher;
private Cipher aesCipher;
private SecretKey sKey1;
private FileInputStream fis;
private FileOutputStream fos;
private static final String keyGeneratorSpec = "AES";
private byte[] block = new byte[4096];
static final org.apache.log4j.Logger logger = org.apache.log4j.Logger.getLogger(FileEncryptionClass.class);
public FileEncryptionClass() {
DOMConfigurator.configure(LogClass.getLogFilePath());
```

```
logger.info("Entered "+this.getClass().getName());
   }

public void encryptyData(File fileInput, File fileOutput) throws NoSuchAlgorithmException,
        NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException,
        BadPaddingException, IOException {

fis = new FileInputStream(fileInput);
fos = new FileOutputStream(fileOutput);
logger.info(" Encrypting file "+fileInput.getName());
     KeyGenerator keyGenS = KeyGenerator.getInstance(keyGeneratorSpec);
keyGenS.init(128);
     //sKey1 = keyGenS.generateKey();
setsKey1(keyGenS.generateKey());
System.err.println("KEY IS " + new BigInteger(getsKey1().getEncoded()).toString(16));
aesCipher = Cipher.getInstance(keyGeneratorSpec);
outputStreamCipher = new CipherOutputStream(fos, aesCipher);

aesCipher.init(Cipher.ENCRYPT_MODE, sKey1);

int i;
while ((i = fis.read(block)) != -1) {
outputStreamCipher.write(block, 0, i);
     }
fos.close();
fis.close();
logger.info("File encryption successful.");
if (fileInput.exists()) {
fileInput.delete();
     }
     //store encryption key
setEncryptionKey(new BigInteger(getsKey1().getEncoded()).toString(16));
setFileName(fileOutput.getName());

writeToXmlFile();
logger.info("saved file encryption key successfully.");
   }

public void decryptData(String fileToAccessPath, String fileToOutputPath) throws IllegalBlockSizeException,
        BadPaddingException, InvalidKeyException, FileNotFoundException, IOException,
NoSuchAlgorithmException, NoSuchPaddingException {
     // new BigInteger(getEncryptionKey(),16).toByteArray();

     File fileInput = new File(fileToAccessPath);
     File fileOutput = new File(fileToOutputPath);

setFileName(fileInput.getName());
getFileEncryptionKey();
     System.out.println(" File name to decrypty " + getFileName() + " key is " + getEncryptionKey() + " decrypt "
+ new BigInteger(getEncryptionKey(), 16).toByteArray().toString());

aesCipher = Cipher.getInstance(keyGeneratorSpec);

aesCipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(new BigInteger(getEncryptionKey(),
16).toByteArray(), keyGeneratorSpec));

fis = new FileInputStream(fileInput);
```

```
fos = new FileOutputStream(fileOutput);

inputStreamCipher = new CipherInputStream(fis, aesCipher);
int i = 0;

while ((i = inputStreamCipher.read(block)) != -1) {
fos.write(block, 0, i);
      }
fos.close();
fis.close();
inputStreamCipher.close();

return;
   }

public SecretKey getsKey1() {
return sKey1;
   }

public void setsKey1(SecretKey sKey1) {
      this.sKey1 = sKey1;
   }

public static void main(String... args) {
   }
}
```

Appendix 4 - Source Code for the Dynamic Generation of secret keys.

Shown below is the source code for the dynamic generation of secret keys based on java prototype:

```
/* ** * code used to generate Secret Keys****/
 To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package com.JSFClasses.kigaita.util;

import com.JSFManagedBeans.kigaita.UploadsManagedBean;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.DocumentHelper;
import org.dom4j.Element;
import org.dom4j.io.SAXReader;
import org.dom4j.io.XMLWriter;

/**
 *
 * @author Kigaita
 */
class XmlUtil {

   //creating document

private  Document document=DocumentHelper.createDocument();
private static  HashMap<String,String> xmlContents=new HashMap<String,String>();
protected String fileName;
protected String encryptionKey;
private static final String xmlFile =new
UploadsManagedBean().getUrlBase()+"\\uploads\\xml\\encryptedFiles.xml";
   /**
    * @param args the command line arguments
    */
public static void main(String[] args) {
      // TODO code application logic here
new  XmlUtil().getXmlFileContents();

   }
protected void getFileEncryptionKey(){
    // this.fileName=fileName;
getXmlFileContents();
System.out.println(" FILE FOUND IS "+this.fileName+" ENCRYPTION IS "+xmlContents.get(this.fileName));

setEncryptionKey(xmlContents.get(fileName));
return ;
  }

protected void writeToXmlFile(){
```

```
System.err.println("PATH TO FILES "+xmlFile);
      //creating element
     Element security=null;
      File xmlFileExists=new File(xmlFile);
if(xmlFileExists.exists()){
System.out.println(" File exists");
try{
document=new SAXReader().read(xmlFile);
security=document.getRootElement();
          }
catch(DocumentException de){
de.printStackTrace();
          }
      }
else{
security=document.addElement("Security");
      }
      //creating element and setting attribute &#!
      Element file=security.addElement("fileName");
file.addAttribute("Name", fileName);

      //creating other elements
      Element encryption=file.addElement("encryptionKey");
encryption.addText(encryptionKey);


      //writing contents to xml file

try{
         XMLWriter xmlWriter=new XMLWriter(
new FileWriter(
new File(xmlFile)
                    )
                );

xmlWriter.write(document);
xmlWriter.close();
System.out.println("Writing file complete");
      }
catch(IOException ioe){
ioe.printStackTrace();
      }

   }

private   void getXmlFileContents() {
System.err.println("In readxml");
System.err.println("PATH TO FILES "+xmlFile);
xmlContents=new HashMap<String,String>();

   //get the saxreader object
      //SAXReader saxReader=new SAXReader();
   //get xml doc
try{
document=new SAXReader().read(xmlFile);
System.out.println("Exception passed");
     }
```

```
catch(DocumentException de){
de.printStackTrace();
      }
    //get all root atts
      Element root=document.getRootElement();


       ArrayList <String> tempAl=null;

for(Iterator i =root.elementIterator();i.hasNext();){
        Element elementFileName=(Element)i.next();
       // System.out.println(" ELEMENT AT START "+elementFileName.attributeValue("Name").toString());
       Element s=(Element)elementFileName.elementIterator().next();
xmlContents.put(elementFileName.attributeValue("Name").toString(), s.getData().toString());
//        System.out.println(" ELEMENT RETURNED "+s.getData().toString()+" OR
"+xmlContents.get(elementFileName.getStringValue()));
//        System.err.println(" FILENAME IS "+elementFileName.getStringValue()+" OR "+fileName+" OR
"+s.getData().toString());

     }

   }

public String getFileName() {
return fileName;
   }

public void setFileName(String fileName) {
      this.fileName = fileName;
   }

public String getEncryptionKey() {
return encryptionKey;
   }

public void setEncryptionKey(String encryptionKey) {
      this.encryptionKey = encryptionKey;
   }

}
```

Appendix 5 - Source Code for storing the generated keys.

Shown below is the source code for the storage of generated secret keys.

Encryption Keys

```
<?xml version="1.0" encoding="UTF-8"?>
<Security><fileName
Name="20143901_3_7967309251_test1.xls"><encryptionKey>6d9b0d8b76dc5cc45078525e81a7fc71</encryp
tionKey></fileName><fileName
Name="20142002_3_15661980371_test1.xls"><encryptionKey>40556ae55237eb02ddd6359c94333bdf</encr
yptionKey></fileName><fileName Name="20144827_3_12970038171_templatefile.xls"><encryptionKey>-
25eadf654198e30c3db167826a115454</encryptionKey></fileName></Security>
```

Appendix 6- Pseudo code for the Prototype

Shown below is the source code for the Pseudo code of the prototype.

```
public class File Encryption Class extends Xml Util {

privateCipherOutputStreamoutputStreamCipher;
privateCipherInputStreaminputStreamCipher;
private Cipher aesCipher;
privateSecretKeysKey1;
privateFileInputStreamfis;
privateFileOutputStreamfos;
private static final String keyGeneratorSpec = "AES";
private byte[] block = new byte[4096];

public void encryptyData(File fileInput, File fileOutput) throws NoSuchAlgorithmException,
NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException,
BadPaddingException, IOException {

fis = new FileInputStream(fileInput);
fos = new FileOutputStream(fileOutput);

KeyGeneratorkeyGenS = KeyGenerator.getInstance(keyGeneratorSpec);
keyGenS.init(128);
      //sKey1 = keyGenS.generateKey();
setsKey1(keyGenS.generateKey());
     // System.err.println("KEY IS " + new BigInteger(getsKey1().getEncoded()).toString(16));
aesCipher = Cipher.getInstance(keyGeneratorSpec);
outputStreamCipher = new CipherOutputStream(fos, aesCipher);

aesCipher.init(Cipher.ENCRYPT_MODE, sKey1);

inti;
while ((i = fis.read(block)) != -1) {
outputStreamCipher.write(block, 0, i);
      }
fos.close();
fis.close();
if (fileInput.exists()) {
fileInput.delete();
      }
      //store encryption key
setEncryptionKey(new BigInteger(getsKey1().getEncoded()).toString(16));
setFileName(fileOutput.getName());
writeToXmlFile();
   }

public void decryptData(String fileToAccessPath, String fileToOutputPath) throws IllegalBlockSizeException,
BadPaddingException, InvalidKeyException, FileNotFoundException, IOException, NoSuchAlgorithmException,
NoSuchPaddingException {
      // new BigInteger(getEncryptionKey(),16).toByteArray();

      File fileInput = new File(fileToAccessPath);
      File fileOutput = new File(fileToOutputPath);

setFileName(fileInput.getName());
```

```
getFileEncryptionKey();
      // System.out.println(" File name to decrypty " + getFileName() + " key is " + getEncryptionKey() + " decrypt
" + new BigInteger(getEncryptionKey(), 16).toByteArray().toString());

aesCipher= Cipher.getInstance(keyGeneratorSpec);

aesCipher.init(Cipher.DECRYPT_MODE,       new       SecretKeySpec(new       BigInteger(getEncryptionKey(),
16).toByteArray(), keyGeneratorSpec));

fis = new FileInputStream(fileInput);
fos = new FileOutputStream(fileOutput);

inputStreamCipher = new CipherInputStream(fis, aesCipher);
inti = 0;

while ((i = inputStreamCipher.read(block)) != -1) {
fos.write(block, 0, i);
      }
fos.close();
fis.close();
inputStreamCipher.close();

return;
   }

publicSecretKey getsKey1() {
returnsKey1;
   }

public void setsKey1(SecretKey sKey1) {
      this.sKey1 = sKey1;
   }

public static void main(String... args) {
   }
```

Appendix 7 – Generated Security Logs

Shown below are the generated Security Logs of the prototype:

07 Apr 2014 18:52:53,086 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - Entered Login page with user onyango
07 Apr 2014 18:52:53,086 [http-thread-pool-8080(3)] ERROR login.LoginManagedBean - User onyango could not be logged in. \n Password and Username combination error.
07 Apr 2014 18:52:53,088 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - Exiting the logging functionality
07 Apr 2014 18:53:05,983 [http-thread-pool-8080(4)] INFO login.LoginManagedBean - Entered Login page with user john
07 Apr 2014 18:53:05,984 [http-thread-pool-8080(4)] ERROR login.LoginManagedBean - User john could not be logged in. \n Password and Username combination error.
07 Apr 2014 18:53:05,984 [http-thread-pool-8080(4)] INFO login.LoginManagedBean - Exiting the logging functionality
07 Apr 2014 18:53:07,045 [http-thread-pool-8080(5)] INFO login.LoginManagedBean - Entered Login page with user john
07 Apr 2014 18:53:07,046 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - User john could not be logged in. \n Password and Username combination error.
07 Apr 2014 18:53:07,047 [http-thread-pool-8080(5)] INFO login.LoginManagedBean - Exiting the logging functionality
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
07 Apr 2014 18:53:08,439 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - Entered Login page with user john
07 Apr 2014 18:53:08,440 [http-thread-pool-8080(1)] ERROR login.LoginManagedBean - User john could not be logged in. \n Password and Username combination error.
07 Apr 2014 18:53:08,441 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - Exiting the logging functionality
07 Apr 2014 18:53:18,042 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - Entered Login page with user eunice
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
07 Apr 2014 18:53:18,043 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - USER Mrs Eunice (eunice) logged in successfully
07 Apr 2014 18:53:18,044 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - Redirecting user eunice to profile page
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
07 Apr 2014 18:53:18,045 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - Exiting the logging functionality
08 Apr 2014 13:44:37,765 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - Entered Login page with user eunice
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
08 Apr 2014 13:44:37,813 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - USER Mrs Eunice (eunice) logged in successfully
08 Apr 2014 13:44:37,813 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - Redirecting user eunice to profile page
08 Apr 2014 13:44:37,818 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - Exiting the logging functionalit
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.

08 Apr 2014 13:49:14,707 [http-thread-pool-8080(3)] INFO com.JSFClasses.kigaita.UploadsController - Entered the uploadsController class
08 Apr 2014 15:39:58,515 [http-thread-pool-8080(2)] INFO login.LoginManagedBean - User is not logged in. Redirecting to login page
08 Apr 2014 15:42:10,283 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - Entered Login page with user eunice
08 Apr 2014 15:42:10,285 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - USER Mrs Eunice (eunice) logged in successfully
08 Apr 2014 15:42:10,286 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - Redirecting user eunice to profile page
08 Apr 2014 15:42:10,288 [http-thread-pool-8080(3)] INFO login.LoginManagedBean - Exiting the logging functionality
08 Apr 2014 15:42:47,412 [http-thread-pool-8080(2)] INFO com.JSFClasses.kigaita.UploadsController - Entered the uploadsController class
08 Apr 2014 15:42:47,413 [http-thread-pool-8080(2)] INFO com.JSFClasses.kigaita.UploadsController - Downloading templatefile.xls.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
08 Apr 2014 15:42:47,480 [http-thread-pool-8080(2)] INFO com.JSFClasses.kigaita.UploadsController - TemplateFile.xls downloaded successfully.
08 Apr 2014 15:44:26,494 [http-thread-pool-8080(3)] INFO com.JSFClasses.kigaita.InformationController - Entered the informationController class
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
08 Apr 2014 15:45:50,048 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.UnitController - Entered the unitController class
08 Apr 2014 15:46:17,894 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.CampusController - Entered the campusController class
09 Apr 2014 10:53:19,690 [http-thread-pool-8080(4)] INFO login.LoginManagedBean - Entered Login page with user eunice
09 Apr 2014 10:53:19,693 [http-thread-pool-8080(4)] INFO login.LoginManagedBean - USER Mrs Eunice (eunice) logged in successfully
09 Apr 2014 10:53:19,694 [http-thread-pool-8080(4)] INFO login.LoginManagedBean - Redirecting user eunice to profile page
09 Apr 2014 10:53:19,695 [http-thread-pool-8080(4)] INFO login.LoginManagedBean - Exiting the logging functionality
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 10:58:05,536 [http-thread-pool-8080(2)] INFO login.LoginManagedBean - Entered Login page with user eunice
09 Apr 2014 10:58:05,539 [http-thread-pool-8080(2)] INFO login.LoginManagedBean - USER Mrs Eunice (eunice) logged in successfully
09 Apr 2014 10:58:05,540 [http-thread-pool-8080(2)] INFO login.LoginManagedBean - Redirecting user eunice to profile page
09 Apr 2014 10:58:05,540 [http-thread-pool-8080(2)] INFO login.LoginManagedBean - Exiting the logging functionality
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 10:58:24,885 [http-thread-pool-8080(3)] INFO com.JSFClasses.kigaita.UploadsController - Entered the uploadsController class

09 Apr 2014 10:58:35,181 [http-thread-pool-8080(5)] INFO com.JSFClasses.kigaita.CourseController - Entered the courseController class
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 10:59:46,035 [http-thread-pool-8080(2)] INFO com.JSFClasses.kigaita.UploadsController - Downloading templatefile.xls.
09 Apr 2014 10:59:46,047 [http-thread-pool-8080(2)] INFO com.JSFClasses.kigaita.UploadsController - TemplateFile.xls downloaded successfully.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 11:02:04,051 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.util.FileEncryptionClass - Entered com.JSFClasses.kigaita.util.FileEncryptionClass
09 Apr 2014 11:02:04,053 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.util.FileEncryptionClass - Encrypting file templatefile (7).xls
09 Apr 2014 11:02:05,677 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.util.FileEncryptionClass - File encryption successful.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 11:02:13,154 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.util.FileEncryptionClass - saved file encryption key successfully.
09 Apr 2014 11:02:13,155 [http-thread-pool-8080(1)] INFO com.JSFManagedBeans.kigaita.UploadsManagedBean - File org.primefaces.model.DefaultUploadedFile@19ea79f uploaded to the server and encrypted successfully
09 Apr 2014 12:20:29,813 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - Entered Login page with user eunice
09 Apr 2014 12:20:29,866 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - USER Mrs Eunice (eunice) logged in successfully
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:20:29,866 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - Redirecting user eunice to profile page
09 Apr 2014 12:20:29,867 [http-thread-pool-8080(1)] INFO login.LoginManagedBean - Exiting the logging functionality
09 Apr 2014 12:20:50,533 [http-thread-pool-8080(5)] INFO com.JSFClasses.kigaita.UploadsController - Entered the uploadsController class
09 Apr 2014 12:20:56,462 [http-thread-pool-8080(2)] INFO com.JSFClasses.kigaita.UploadsController - Downloading templatefile.xls.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:20:56,472 [http-thread-pool-8080(2)] INFO com.JSFClasses.kigaita.UploadsController - TemplateFile.xls downloaded successfully.
09 Apr 2014 12:21:04,240 [http-thread-pool-8080(3)] INFO com.JSFClasses.kigaita.UploadsController - Downloading templatefile.xls.
09 Apr 2014 12:21:04,241 [http-thread-pool-8080(3)] INFO com.JSFClasses.kigaita.UploadsController - TemplateFile.xls downloaded successfully.
09 Apr 2014 12:21:04,299 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.UploadsController - Downloading templatefile.xls.
09 Apr 2014 12:21:04,299 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.UploadsController - TemplateFile.xls downloaded successfully.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:35:33,613 [http-thread-pool-8080(3)] INFO com.JSFClasses.kigaita.InformationController - Entered the informationController class
09 Apr 2014 12:35:35,995 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.ExaminationsController - Entered the examinationsController class
09 Apr 2014 12:35:39,007 [http-thread-pool-8080(4)] INFO com.JSFClasses.kigaita.UserController - Entered the userController class

09 Apr 2014 12:35:41,791 [http-thread-pool-8080(1)] INFO com.JSFClasses.kigaita.StudentController - Entered the studentController class
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:36:00,699 [http-thread-pool-8080(2)] INFO com.JSFClasses.kigaita.UnitController - Entered the unitController class
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:36:15,551 [http-thread-pool-8080(5)] INFO login.LoginManagedBean - Entered Login page with user eunice
09 Apr 2014 12:36:15,553 [http-thread-pool-8080(5)] INFO login.LoginManagedBean - USER Mrs Eunice (eunice) logged in successfully
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] INFO login.LoginManagedBean - Redirecting user eunice to profile page
09 Apr 2014 12:36:15,554 [http-thread-pool-8080(5)] ERROR login.LoginManagedBean - failed decryption attempt.

.