**IMPLEMENTATION OF A NEW SECURITY ALGORITHM USING HYBRID CRYPTOGRAPHY FOR PASSWORD PROTECTION**

*A practice school report submitted to*

**MANIPAL UNIVERSITY**

*For Partial Fulfilment of the Requirement for the*

*Award of the Degree*

*Of*

**BACHELOR OF ENGINEERING**

*In*

**COMPUTER SCIENCE AND ENGINEERING**

*By*

**PREETHISH SHETTY**

**100905013**

*Under the Guidance of*

**DR. KRISHNAMOORTHI MAKKITHAYA**

PROFESSOR

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

****

**MANIPAL INSTITUTE OF TECHNOLOGY**

(A constituent Institute of MANIPAL UNIVERSITY)

**MANIPAL - 576 104, KARNATAKA, I NDIA**

**2014**

**IMPLEMENTATION OF A NEW SECURITY ALGORITHM USING HYBRID CRYPTOGRAPHY FOR PASSWORD PROTECTION**

*A practice school report submitted to*

**MANIPAL UNIVERSITY**

*For Partial Fulfilment of the Requirement for the*

*Award of the Degree*

*Of*

**BACHELOR OF ENGINEERING**

*In*

**COMPUTER SCIENCE AND ENGINEERING**

*By*

**PREETHISH SHETTY**

**100905013**

*Under the Guidance of*

**DR. KRISHNAMOORTHI MAKKITHAYA**

PROFESSOR

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

****

**MANIPAL INSTITUTE OF TECHNOLOGY**

(A constituent Institute of MANIPAL UNIVERSITY)

**MANIPAL - 576 104, KARNATAKA, I NDIA**

**2014**

**MANIPAL INSTITUTE OF TECHNOLOGY**

(A constituent Institute of MANIPAL UNIVERSITY)

**MANIPAL - 576 104, KARNATAKA, I NDIA**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CERTIFICATE**

This is to certify that the project work of

**PREETHISH SHETTY**

on the project ***Implementation of a New Security Algorithm using Hybrid Cryptography for Password Protection*** is a record of the bonafide work carried out in partial fulfilment of the requirements for awarding the degree of **Bachelor of Engineering** in **Computer Science and Engineering** discipline in **Manipal Institute of Technology** under **Manipal University**, Manipal during the academic year 2014.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Dr. RENUKA A**

**Head of the Department**

**Dept. of Computer Science & Engg**.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**DR. KRISHNAMOORTHI MAKKITHAYA**

**Professor, Project Guide (Internal)**

**Dept. of Computer Science & Engg.**

"The difference between the impossible and the possible lies in a man's determination."

**-- Tommy Lasorda**

**ACKNOWLEDEGEMENTS**

I take this opportunity to express my deepest appreciation to all those who provided me the opportunity to complete my final year project.  I extend my special gratitude to my Project Internal Guide **Dr Krishnamoorthi Makkithaya, (*Professor, Department of Computer Science & Engg*)**whose contribution in creating an educative training atmosphere, helped me complete the project in a satisfactory manner. His valuable guidance and inputs helped me analyze and implement the project better.

I would also like to thank **Mr. Govardhan Hegde,** who was the faculty member under whom I took this project up. He has been a beacon of constant support and encouraged me to implement innovative ideas and techniques in the project. Furthermore I would also like to acknowledge with much appreciation the crucial role of the MIT and Computer Science Department staff and lab in-charges for their support and help in facilitating with the use of all resources required to complete the project.

I also extend my deep sense of gratitude to **Dr. Vinod V Thomas** (Director, MIT, Manipal) and **Dr. Renuka A** (HOD, Department of Computer Science Engg. ) for making us realize the importance of the Final Year Project process as part of the course curriculum. Last but not the least I place a deep sense of gratitude to my family members and my friends who have been constant source of inspiration during the course of this project work.

Preethish Shetty

i

**ABSTRACT**

**Name of the faculty:** Govardhan Hegde

**Area of Computer Science:** Computer Networks

Project title:

*"Implementation of a new security algorithm using hybrid cryptography for password protection"*

This project implements a hybrid cryptographic encryption scheme on both client and server machines to allow encrypted password protection. Although there are many Java applications for password protection, only few use MD5 on the server-side, and as a result the password still travels over the internet as plain text. All server-side only schemes are completely open to packet-sniffing. With this scheme, the browser encrypts the password on the client's machine making simple packet-sniffing more difficult.

Client-side scripting is done using JavaScript and server-side scripting is done with JSP and Java. We make use of Microsoft SQL Server for databases. This is a one-way encryption scheme: the password is never transmitted or stored as clear text on the client or server, and thus cannot be recovered. In addition, the encryption scheme is a hybrid architecture combining both symmetric and asymmetric key cryptography. This hybrid ensures a more robust and lightweight encryption algorithm that provides integrity, authentication, confidentiality and reliability. A hybrid encryption algorithm proposed is more secure in comparison to the existing encryption standards and is less prone to attacks.

ii

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **INDEX** | | | | | **Page** |
|  | | **ACKNOWLEDGMENT** | | | **i** |
|  | | **ABSTRACT** | | | **ii** |
|  | | **LIST OF TABLES** | | | **vii** |
|  | | **LIST OF FIGURES** | | | **vii** |
|  | | **ABBREVATIONS** | | | **viii** |
|  | | **NOTATIONS** | | | **Ix** |
| **1.** |  | **INTRODUCTION** | | | **1-3** |
|  | 1. | General Introduction to the Topic | | | 1 |
|  | 2. | Organization | | | 1 |
|  | 3. | Area of Computer Science | | | 2 |
|  |  | 1. | | Computer and Network Security | 2 |
|  |  | 2. | | Services, Mechanisms and Attacks | 2 |
|  |  | 3. | | Cryptography | 2 |
|  | 4. | Hardware and Software Requirements | | | **3** |
|  |  | 1. | | Hardware Requirement | 3 |
|  |  | 2. | | Software Requirement | 3 |
| **2.** |  | **LITERATURE SURVEY** | | | **4-6** |
| **3.** |  | **RESEARCH QUESTION/PROBLEM DEFINITION** | | | **7** |
| **4.** |  | **OBJECTIVES** | | | **8** |
| **5.** |  | **BACKGROUND** | | | **9** |
| **6.** |  | **EXISTING SOLUTIONS** | | | **10** |
|  | 1. | Lack of Client Side Encryption | | | 10 |
|  | 2. | Weakness of Encryption Techniques | | | 10 |
| **7.** |  | **NEW SUGGESTIONS** | | | **11-13** |
|  | 1. | Client Side Encryption | | | 11 |
|  | 2. | Hybrid Cryptography Encryption Algorithm | | | 11 |
|  |  | 1. | | Working of Proposed Hybrid Encryption | 12 |
|  |  | 2. | | Advantages of using ECC generated Key | 12 |
|  |  | 3. | | NIST Recommended Key Sizes | 13 |
| **8.** |  | **METHODOLOGY** | | | **14-35** |
|  | 1. | Application of Client Side Encryption | | | 14 |
|  | 2. | Hybrid Algorithm Architecture | | | 15-16 |
|  | 3. | MD5 | | | 16-19 |
|  |  | 1. | | Input Pre-processing | 16 |
|  |  | 2. | | Compression Function for 512 bit blocks | 17 |
|  |  | 3 | | One Round of the Compression Function | 18-19 |
|  | 4. | Elliptical Curve Cryptography | | | 19-22 |
|  |  | 1. | | Introduction and History of ECC | 19 |
|  |  | 2. | | Working of ECC | 20 |
|  |  | 3. | | ECC in Public Key Cryptography System | 21 |
|  |  | 4. | | Elliptical Curve Diffe Hellman | 21-22 |
|  |  | 5. | | Advantage of ECC | 22 |
|  | 5. | RSA | | | 23-25 |
|  |  | 1. | | Operation of RSA | 23 |
|  |  |  | 1. | Key Generation | 23-24 |
|  |  |  | 2. | Encryption | 25 |
|  |  |  | 3. | Decryption | 25 |
|  |  | 2. | | Factorization Problem | 25 |
|  | 6. | Java Server Pages | | | 26-31 |
|  |  | 1. | | Overview | 26 |
|  |  | 2. | | Java Servlets | 26-27 |
|  |  | 3. | | Life Cycle of JSP | 27-29 |
|  |  | 4. | | Overview of JSP | 29-30 |
|  |  | 5. | | JSP Compiler | 31 |
|  | 7. | Javascript | | | 31- 34 |
|  |  | 1. | | Overview | 31-32 |
|  |  | 2. | | Use in Web Page | 32-34 |
|  |  | 3. | | Use outside Web Pages | 34 |
|  |  | 4. | | JavaScript vs Java | 34 |
|  | 8. | Apache Tomcat | | | 35 |
|  | 9. | MySQL Database | | | 35 |
| **9.** |  | **IMPLEMENTATION DETAILS** | | | **36-47** |
|  | 1. | Implementation of the Hybrid Architecture | | | 36-39 |
|  |  | 1. | | ECC - Key Pair Generation | 36 |
|  |  | 2. | | Key Agreement | 36-37 |
|  |  | 3. | | RSA | 37-38 |
|  |  | 4. | | Hybrid Architecture | 39 |
|  | 2. | Message Digest 5 | | | 40-44 |
|  |  | 1. | | Initialization | 40 |
|  |  | 2. | | Compression Function | 40-41 |
|  |  | 3. | | Padding and Block Formation | 41 |
|  |  | 4. | | Trace of Padding | 42 |
|  |  | 5. | | Main MD5 Processing Block | 43-44 |
|  | 3. | Client Server Login | | | 44-47 |
|  |  | 1. | | Client Login Form | 44-45 |
|  |  | 2. | | MySQL Database | 46-47 |
| **10.** |  | **EXPERIMENTAL RESULTS AND ANALYSIS OF RESULTS** | | | **48-53** |
|  | 1. | Results of different Algorithm Implementation | | | 48-51 |
|  |  | 1. | | MD5 | 48 |
|  |  | 2. | | ECDH Key Agreement | 48-49 |
|  |  | 3. | | RSA | 49-50 |
|  |  | 4. | | Hybrid Architecture | 50-51 |
|  | 2. | Client - Server Login | | | 51-53 |
|  |  | 1. | | Client Login Form | 51-52 |
|  |  | 2. | | Client Login Success Page | 52 |
|  |  | 3. | | Client Login Failure Page | 52-53 |
| **11.** |  | **CONCLUSION** | | | 54 |
| **12.** |  | **REFERENCES** | | | **55** |
| **13.** |  | **PROJECT DETAILS** | | | **56** |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Table** | **Text** | **Page** |
| 7.1 | NIST Recommended Key Sizes | 13 |
| 8.1 | Primitive Function | 19 |
|  |  |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure** | **Text** | **Page** |
| 8.1 | Hybrid Algorithm Architecture | 15 |
| 8.2 | MD5 Processing a Single 512 Bit Block | 17 |
| 8.3 | Elementary MD5 Operation | 18 |
| 8.4 | Life of a JSP File | 29 |
| 8.5 | JSP Model 2 Architecture | 30 |
| 9.1 | MySQL Server End Configuration | 46 |
| 9.2 | MySQL Client End Configuration | 46 |
| 9.3 | MySQL Database Entries | 47 |
| 10.1 | User Login Page | 52 |
| 10.2 | Successful Login Page | 52 |
| 10.3 | Failure of Login Page | 53 |

vii

**ABBREVIATIONS**

|  |  |
| --- | --- |
| **ECC** | Elliptical Curve Cryptography |
| **ECDH** | Elliptical Curve Diffe Hellman |
| **MD5** | Message Digest 5 |
| **SHA** | Secure Hash Algorithm |
| **ECDLP** | Elliptic Curve Discrete Logarithm |
| **GNFS** | General Number Field Sieve |
| **ASP** | Active Server Page |
| **JSP** | Java Server Page |
| **JVM** | Java Virtual Machine |
| **HTML** | Hyper Text Markup Language |
| **JS** | JavaScript |

viii

**NOTATIONS**

Message Digest 5 Implementation :

|  |  |
| --- | --- |
| **a** | First Buffer Entry |
| **b** | Second Buffer Entry |
| **c** | Third Buffer Entry |
| **d** | Fourth Buffer Entry |
| **X[k]** | Current Bit of the word being processed |
| **T[i]** | ith word of the T Matrix |
| **+** | Addition modulo 2^32 |
| **g** | Either one of the primitive function F,G,H,I |
| **<<<s** | Circular Left Shift of the 32 bit argument by s bits |

Encryption Standards :

|  |  |
| --- | --- |
| **e** | Encryption Key |
| **d** | Decryption Key |
| **(e,N)** | Public Key |
| **(d,N)** | Private Key |
| **G** | Generator Point |
| **p** | Prime number |
| **q** | Prime Number |

ix

**CHAPTER 1**

**INTRODUCTION**

* 1. **GENERAL INTRODUCTION TO THE TOPIC :**

The digital age we are currently living in is a password-driven world, where between 4 and 20 characters make the difference in communicating with friends, accessing of data and information or making online transactions. This information is a potential goldmine for those intending to use it for the wrong purpose.

The crux of the project is in strengthening the strength of passwords in a twofold manner. The project is aimed at implementing a new security algorithm, on both client and server end, and one that uses a hybrid cryptographic architecture. The existing system is prone to collision attacks and can be compromised by attackers. In order to strengthen the system, hybrid architecture is proposed that is more resistant to attack and hence can provide a secure form of information exchange between connected computers. This new security algorithm is designed using a combination of both symmetric and asymmetric cryptographic techniques. This can be achieved by the combinatorial effect of Dual RSA, Elliptic Curve Cryptography implemented by **ECDH** andHash algorithm implemented by **MD5**.

**1.2 ORGANIZATION**

This project was done in Manipal Institute of Technology, a constituent institution of the Manipal University. The project duration is 4 months and is being done under the guidance of Dr. Krishnamoorthi Makkithaya, Professor in the Department of Computer Science and Engineering in MIT.

**1.3 AREA OF COMPUTER SCIENCE :**

**1.3.1 Computer and Network Security:**

With the advent of computers, the need for automated tools for protecting files and other information stored on the computer became evident. The generic name for the collections of tools designed to protect data and to thwart attackers is computer security.

Similarly the introduction of distributed systems and the use of networks for carrying data between terminal user and computers or computer and computer has led to the need of measure to protect data namely network security.

**1.3.2 Services, Mechanism and Attacks:**

Security Attack: Any action that compromises the security of information owned by the organization.

Security Mechanism: A mechanism that is designed to detect, prevent or recover from a security attack.

Security Service: A service that enhances the security of the data processing systems and the information transfers of an organization. The services are intended to counter security attacks, and they make use of one or more security mechanism to provide the service.

**1.3.3 Cryptography:**

Cryptography is the science which uses mathematics to encrypt and decrypt data. This science enables you to store sensitive information or transmit it across insecure networks so that it cannot be intercepted and be read by anyone except the intended recipient.

Cryptanalysis is the science of analyzing and breaking secure communication. It involves a combination of analytical reasoning, application of mathematical tools, pattern finding and determination. They are also called “Attackers”.

A cryptographic algorithm, or cipher, works in combination with a key (number, phrase, word) to encrypt the plaintext. The same plain text encrypts to different cipher text with different keys. The key feature being two things: the strength of the algorithm and secrecy of the key.

**1.4 Hardware and Software Requirements:**

**1.4.1 Hardware Requirements (Minimum):**

Central Processing Unit : Pentium 4, 2.66 GHZ

Hard Disk Capacity : 10 GB

Memory : 256 MB RAM

Peripherals : Monitor, Keyboard, Mouse

**1.4.2 Software Requirements (Minimum):**

Server Side

Front End : Eclipse

Back End : Microsoft SQL Server

Client Side

Operating System : Windows XP

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 Introduction:**

The project was conceptualized after researching the need for more stringent network security measures. With the rise of e-commerce and e-business over the years, it is essential that user data and user identities are validated to avoid incurring losses and data theft. This situation warranted for more secure data and a reliable communication channel. With these aspects in the backdrop, the research was conducted and lead to the finding of 2 main papers to corroborate the research interest.

The paper, **“*Research for the Application and Safety of MD5 Algorithm in Password Authentication****” by Xiaoling Zheng and Jidong Jin***,** addressed the first issue which dealt with some of the most common security problems faced with the internet. The major concern being: monitoring of network data by attackers to gain access to sensitive data and user credentials. The paper proposed a more secure communication means by suggesting both Client and Server Side Encryption, which forms the first principal feature of this project.

With client side encryption, the user data is transmitted across the transmission medium in an encrypted form preventing attackers from accessing the information. This makes for a more secure form of communication. The encryption standard proposed here was MD5 or Message Digest 5. This is a Hashing Based Algorithm developed jointly by MIT Computer Science Laboratory and Ronal L. Rivest from RSA Data Security Inc. While the encryption on client side ensured the data never travelled as plaintext, MD5 has over the years been compromised and collision attacks have been designed to weaken the encryption standard. Hence the research effort was directed to find another suitable encryption that was more reliant and less prone to be compromised by attackers.

The paper, *“****Design of New Security Algorithm using Hybrid Cryptography Architecture****” by Manali J Dubal, Mahesh T R, Pinaki A Ghosh,* proposed the design of a hybrid cryptography algorithm which formed the second principle of the project. The current encryption standards in a standalone manner are weak and can be broken. For example, the once renowned hashing algorithm MD5 has been broken and hence can no longer be used for direct encryption as it carries the risk of being compromised. With this in mind, the idea of using a hybrid architecture is based on the principle of using the strong features of each of the encryption standards. The proposed hybrid architecture was a combination of both symmetric key cryptography and asymmetric key cryptography ensuring integrity, confidentiality and authentication. This will be achieved using a combination of Elliptical Curve Cryptography implemented by ECDH, RSA and Hash Algorithm using Message Digest 5.

Wang Yu, Chen Jianhua, He Debiao in their paper, "***A new collision attack on MD5*" & an article by Dario Forte titled "*The Death of MD****5*" highlighted that the hashing based algorithm is no longer resistant to attacks. A stronger and more secure form of encryption was deemed necessary to thwart imminent attacks and loss of data. This paper provided valuable insights into what are the factors that go into breaking an algorithm.

V Gayoso Martinez and L. Hernandez Encinas in their paper,"***Implementing ECC with Java Standard Edition 7***", provided an insight into using the Java platform which includes a cryptographic provider, named SunEC, to implement some of the elliptical curve operations and protocols. ECC is one the best options for protecting sensitive information and most users are unaware of the potential due to lack of information available. The SunEC, cryptographic engine, supports ECC of the shelf. In addition, the paper gave a brief introduction into the mathematic involved with elliptical curves.

# The RFC 1321 - The MD5 Message-Digest Algorithm, by the MIT Laboratory for Computer Science, provides well documented information on Message Digest 5. In addition to detailed working, it provides the set of tested inputs and outputs which was instrumental in deciding the validity of the implemented algorithm.

**CHAPTER 3**

**RESEARCH QUESTION/PROBLEM DEFINITION**

With an ever increasing growth in Internet and a large shift to virtual presence there is a strong need for stronger computer and network security. Password and identity theft are one of the most alarming trends of modern day Internet. In the current scenario regarding password security, there are many Java applications for password protection. However in many cases the password travels over the internet as plain text. All server-side only schemes are completely open to packet sniffing. With passwords being transmitted in plain text it is easy for an attacker to retrieve sensitive information and use it against the users of the transmission medium.

The project aims at strengthening this transmission process and developing a secure communication channel that ensures integrity and data protection. The key idea is to implement a hybrid encryption scheme on both client and server machines. In addition to both side encryptions, the project aims at strengthening the encryption process by implementing a hybrid cryptography algorithm. In addition, the reason for undertaking such hybrid architecture is that it offers better security for a smaller key size which co-relates to higher speeds, lower power consumption, storage efficiency and lesser bandwidth.

**CHAPTER 4**

**OBJECTIVES**

The primary objective of the project is to ensure robustness of Password Protection and implementing secure mechanisms to ensure passwords from being stolen. The project focuses on two aspects namely : Client Side Encryption and Embedding a Hybrid Cryptographic Algorithm to thwart attacks from malicious users.

* The objective of this project is to encrypt passwords on the client side using hybrid cryptography architecture and transmit it through the transmission channel to the server side.
  + In the situation where passwords are sent in plaintext, they can be easily intercepted and used for wrong purposes by the perpetuator.
* The hybrid architecture prevents the password from being intercepted by attackers snooping in on the transmission medium.
* In addition, hybrid architecture is that it offers better security for a smaller key size which co-relates to higher speeds, lower power consumption, storage efficiency and lesser bandwidth.
* The sent and received messages should be identical ensuring data integrity. This is verified using the MD5 hash of the cipher text on the server side for authentication purpose on the server side.
* The encrypted password is then compared with a stored set of hashed passwords indexed by the username and checked if it exists in the database. This is done to ensure that the passwords are never stored in the database in plaintext adding another security layer in the development of the application.

**CHAPTER 5**

**BACKGROUND**

Due the rapid development of Internet, e-commerce and e-business are used more widely and frequently. However, while these applications bring great convenience to our work and lives, they also bring increasingly serious security problems. Identity authentication is an important method to ensure safety of various e-commerce activities, e-government business information and internet information. However, the internet is treacherous and user’s information is easily achieved and copied. Therefore identifying legitimate users and protecting elusive information of sensitive data is of pivotal importance.

2012 was coined as the year of Password Theft with a reported 300% increase in password theft. In comparison to Q1 of 2010 where 9.5 million pieces of personal information was sold illegally, the number has swelled to 12 million in 2012. With a larger number of devices connected to the internet, smart phones as well as computers, there is a strong need for password protection in the realm of computer networks.

Traditional password authentication with basic mode of user name and password faces many security problems:

1. Eavesdrop through user’s host. Trojan horse programs coming from different channels can be activated by user’s carelessness and give the hacker remote access.
2. Monitor through network data. Sniffing tools and analyzing network traffic data using various tools can enable attackers to retrieve valuable information.

The biggest problem of the traditional password authentication method is the user’s name and password is transmitted in plain code. In order to prevent the problems mentioned above threatening computer security, the common resolution is to use an encryption schema at the client side itself to encrypt data before it is sent across in the transmission medium.

**CHAPTER 6**

**EXISTING SOLUTIONS**

**6.1 Lack of Client Side Encryption:**

Many applications use encryption for password protection but at times the transmission medium is vulnerable to attack. Primarily the reason for this is that the passwords are transmitted across the medium in plaintext. When a client enters his credentials or any sensitive data which is sent across the medium in plaintext, it is vulnerable to attacks or interception by those looking out for such information. As authentication information is usually transmitted from the client end to the server via the network, attackers can monitor these data using special tools and abstract user names, passwords and other authentication information through analysis.

**6.2 Weakness of Encryption Techniques:**

With the introduction of a new cryptographic algorithm, cryptanalysts spend hours together to break the algorithm. While an attacker does these for potentially illicit or harmful reasons, there exist another set of cryptanalysts who analyze the algorithm to discover potential flaws and vulnerabilities. In the recent years, there has been considerable effort, and some success in developing cryptanalytic attacks on all forms of encryption algorithms including hash functions. While client side encryption helps reducing the vulnerability of an interception, a weak algorithm can be easily broken by an attacker. For instance, encryption algorithms like MD4 and MD5 were compromised which has led to the advent of SHA and its various improvements.

**CHAPTER 7**

**NEW SUGGESTION**

**7.1 CLIENT SIDE ENCRYPTION:**

The technique to combat the existing system is to ensure an encryption scheme on both client and server machines. Since all server-side only schemes are open to attacks, encryption of user information on the client-side will help achieve cryptographic primitives such as integrity, confidentiality and authentication. The new suggestion is to ensure all forms of communication between the client and server is encrypted. Instead of sending the user credentials and sensitive information as plaintext across the transmission medium, encrypt it with a suitable algorithm and transmit the encrypted text across the transmission channel.

**7.2 HYBRID CRYPTOGRAPHY ENCRYPTION ALGORITHM:**

This can be achieved by using a combinatorial effect of coupling Elliptical Curve Cryptography (ECC) implemented by ECDH & RSA and Hash Algorithms using Message Digest 5 (MD5). This new algorithm has been designed for better security using a combination of both symmetric and asymmetric cryptography techniques. The proposed hybrid is robust and a lightweight protocol that has a smaller key size, thereby reducing processing overload. The benefits of this higher strength per-bit include higher speeds, lower power consumption, bandwidth saving, storage efficiency and smaller certificates.

**7.2.1 Working of Proposed Hybrid Encryption:**

ECC, which implements some elliptical curve operations, is used to generate a key using ECDH. This Key is used to formulate the Private and Public Key used in the RSA encryption algorithm. The ECC generated key and the plaintext is fed into the modified RSA Algorithm where it is encrypted to generate a crypt text. To ensure authentication, a hash is generated of the crypt text using MD5. This crypt text + hash is now transmitted across the medium to the server side. Since the password is now encrypted it is less prone to be attacked by illicit users and attackers.

On the server side, the hash is recomputed for the crypt text using MD5 and compared with the sent hash. Any change in the password during transmission will result in the hash's not complying with each other. This ensures authentication of the password login. Once it has been authenticated, the password is decrypted using the RSA algorithm and the plain text is retrieved. In order to avoid passwords being stored in the server database in plaintext, the concept involved is storing the hashes of the plaintext in the server database. To compare, the plaintext is hashed and checked with the database to see if there is a match. Based on the result, the user is redirected to a Login Success or Login Failure Page.

**7.2.2 Advantages of using ECC generated Key:**

Rather than using the concept of prime numbers and the Factorization Problem as the rationale behind RSA, the use of ECC generated key enables for a more complex key generation that is harder to crack. The ECC is based on what is known as a Elliptic Curve Discrete Logarithm problem (ECDLP). While the RSA relies on the Factoring problem, which can be solved using the General Number Field Sieve (GNFS) algorithm which has a sub exponential time algorithm, the ECC works on ECDLP. The fastest algorithm to solve ECDLP is the Pollard rho algorithm which has an exponential-time algorithm.

**7.2.3 NIST Recommended Key Sizes:**

|  |  |  |
| --- | --- | --- |
| Symmetric Algorithm (bit) | RSA and DH (bit) | ECC (bit) |
| 56 | 512 | 112 |
| 80 | 1024 | 160 |
| 112 | 2048 | 224 |
| 128 | 3072 | 256 |
| 192 | 7680 | 384 |
| 256 | 15360 | 521 |

Table 7.1 : NIST Recommended Key Sizes

**CHAPTER 8**

**METHODOLOGY**

**8.1 Application of Client Side Encryption:**

The premise of security of various internet applications is effective identification and authentication to identities. There are several identity authentication methods realizing the users’ identification and the user name/password is the easiest and most common method, which is to confirm the user validity through password matching. The general method is to build a form storing user information in the database, which should at least contain 3 fields, user name, password and privilege level. As user visits the system, it will verify the match of user name and password with the relevant information stored in the database. In general, user information in the database is transmitted and stored in the form of initial plain code, the security of which is obviously relatively low.

In order to increase the security of password to prevent users’ password leaking, the hybrid algorithm could be used to exchange the content of users’ password. Realization method is to use MD5 algorithm to process the users’ password, and store the changed value in database. The length of the password content is value of 128 bits. When we verify the users’ identifications using signature comparison, same algorithm processes the decrypted password and compares it with the hash value stored in database. If the two values are the same, the user is valid. As the MD5 algorithm is irreversible, the value of information abstract calculated by MD5 cannot get the initial information by inverse algorithm. This method cannot only avoid the system administrator obtaining the password, but also increase the difficulties to decipher password in some degree.

**8.2 Hybrid Algorithm Architecture:**

****

Fig. 8.1 Hybrid Algorithm Architecture

The hybrid algorithm works to integrate both symmetric and asymmetric cryptography techniques. As shown in the figure, the Symmetric Key Cryptographic Techniques such as Elliptical Curve Cryptography and MD5 are used to achieve both confidentiality and Integrity. The Asymmetric Key Cryptography technique uses RSA for authentication.

On the Encryption side, the given plaintext (user password) is encrypted with the help of a key generated by the Elliptic Curve Cryptography ie ECDH. The key generated forms the Private and Public Key & the password forms the Plain Text. The next step is where the plain text and the key are taken by the RSA and generates a cipher text. Simultaneously, the encrypted cipher text is taken through the Message Digest 5 algorithm. Now the generated cipher text and the signature (MD5 hash) can be communicated to the destination through any secured channel.

On the Decryption side, the hash value is first evaluated from the cipher text and integrated. This is compared with the signature, for the verification of the digital signature appended at the end of message. Thereafter, the decryption of cipher text is done by RSA. Hence, the plaintext can be derived and further the MD5 processes it to form the hash value. This hash value is compared with the set of digests stored in the server database to find whether the user is a valid one or not.

**8.3 MD5:**

Message Digest 5 is a hashing based encryption technique. It takes in a variable length message and processes it to form a 128bit output. The entire process consisting of 64 operations grouped into 4 rounds of 16 operations. A separate function is used in each of these 4 rounds. A 128 bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as 4 32 bit registers (A,B,C,D). These registers are initialized as:

A = 67452301

B = EFCDAB89

C = 98BADCFE

D= 10325476

**8.3.1 Input Preprocessing:**

The input provided to the function is divided into 512-bit blocks; the input is padded such that the length is divisible by 512. The padding works as follows: first byte after the input is followed by 0x80 (10000000). The remaining bits are filled with zeroes till the last 8 bytes. In these 8 bytes, the length of the input is stored in a 64-bit integer representation.

**8.3.2 Compression Function for 512 bit block:**

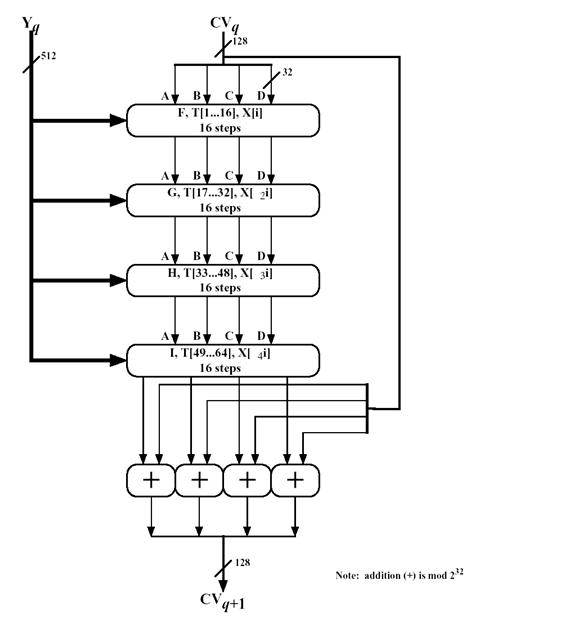


Fig. 8.2 MD5 Processing of a Single 512 Bit Block (MD5 Compression Function)

The four rounds have similar structure but each uses a different primitive logical function referred to as F, G, H and I. Each round takes as input the current 512 bit block and the 128 bit buffer ABCD and updates the content of the buffer. Each round also makes use of one-fourth of a 64 element Table T[1...64], constructed by the sine function. The table provides a randomized set of 32-bit patterns which should eliminate regularities in the input data. The ith element of T, denoted by T[i], is equal to integer part of :

*T*[*i*] <- *2* ^ *32* \* *abs*( *sin*( *i* ) ), where *i* is in radians.

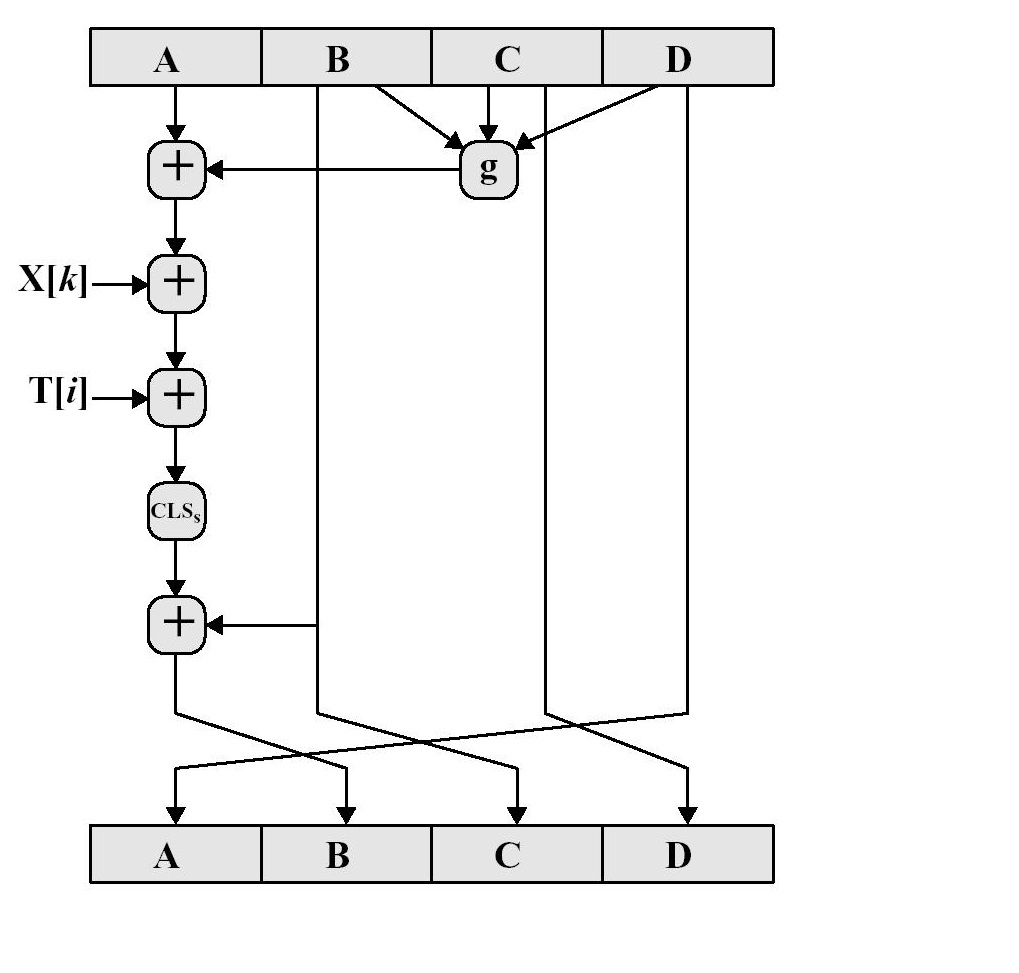
**8.3.3 One Round of the Compression Function:**

Fig. 8.3 Elementary MD5 Operation (Single Step)

Each round of the compression function consists of 16 steps making a total of 64 processing steps for a 512 bit block. Each step is of the form:

*a* <- *b* + (( *a* + *g*(*b,c,d*) + *X*[*k*] + *T*[*i*]) <<< *s* )

where

*a,b,c,d* = the four words of the buffer

*g* = either one of the primitive function F, G, H, I

<<<s = Circular Left Shift of the 32 bit argument by s bits.

*X*[*k*] = the current bit of the word being processed.

*T*[*i*] = The ith word of the T matrix

+ = addition modulo 2^32

One of the 4 primitive logical functions is used for each of the 4 rounds of operation. Each primitive function takes in a 32 bit word as input and produces a 32 bit word output. The functions can be summarized as follows:

|  |  |  |
| --- | --- | --- |
| **Round** | **Primitive Function g** | **g(b,c,d)** |
| 1 | F(b,c,d) | (b&c) | (~b&d) |
| 2 | G(b,c,d) | (b&d) | (c&~d) |
| 3 | H(b,c,d) | b ^ c ^ d |
| 4 | I(b,c,d) | c ^ (b | ~d) |

Table 8.1 : Primitive Function

**8.4 Elliptical Curve Cryptography :**

**8.4.1 Introduction and History of ECC:**

The main problem of conventional public key cryptography systems is that the key size has to be sufficiently large in order to meet the high-level security requirement. This results in lower speed and consumption of more bandwidth. The solution to this comes in the form of Elliptic Curve Cryptography system. In 1985, Neal Koblitz and Victor Miller independently proposed using elliptic curves to design public key cryptographic systems. In the late 1990`s, ECC was standardized by a number of organizations and it started receiving commercial acceptance. Nowadays, it is mainly used in the resource constrained environments, such as ad-hoc wireless networks and mobile networks. \_ There is a tend that conventional public key cryptographic systems are gradually replaced with ECC systems. As computational power evolves, the key size of the conventional systems is required to be increased dramatically.

**8.4.2 Working of ECC :**

While using **ECC**, we deal with various properties of points on curve, and functions. The only aim is to use elliptic curves as an encryption tool which *converts the information*

*m into the point P on the curve E*.

Assuming that the data *m* is already in the integer or any ASCII format, we have a curve

E : y2 = x3+ ax + b (mod p).

This will work only if m3 + am + b is a square modulo p. Since only half of the numbers modulo p are squares, we only have about half a chance of this occurring. Thus, we embed the information **m** into a value that is a square. We assume some value **K** such that 1/2K is an acceptable failure rate for embedding the information into a point on the curve. Also, we make sure that :

For the value of K, (m + 1) K < p.

Now, let xj = mK + j for j = 0, 1, 2, . . . ,K – 1, we compute x3j + axj + b.

The square root yj (mod p) is calculated. If there is a square root, we let our point on E representing m be Pm = (xj, yj). So, for each value of j we have a probability of about 0.5 that xj is a square modulo p. Thus, the probability that no xj is a square is about 1/2K,

which was the acceptable failure rate. In this way, the rate of failure of embedding the information in E decreases.

**8.4.3 ECC in Public Key Cryptography System:**

Elliptic curves are used to construct the public key cryptography system. Like the conventional cryptosystems, once the **Key Pair (*d*, *Q*)** is generated, a variety of cryptosystems such as signature, encryption/decryption, key management system can be set up. Computing ***dP***is denoted as scalar multiplication. It is not only used for the computation of the public key but also for the signature, encryption, and key agreement in the ECC system.

Private Key : **d**is randomly selected from [1,*n*-1], where *n* is integer.

Public Key : ***Q***is computed by ***dP***, where *P,Q* are points on the elliptic curve.

**8.4.4 Elliptical Curve Diffe Hellman (ECDH) :**

Elliptic curve Diffie–Hellman (ECDH) is an anonymous key agreement protocol that allows two parties, each having an elliptic curve public–private key pair, to establish a shared secret over an insecure channel. This shared secret may be directly used as a key, or better yet, to derive another key which can then be used to encrypt subsequent communications using a symmetric key cipher.

Using **ECDH** in the hybrid algorithm generates the key which is far secured than any other algorithm. The key generated is kept as a shared key between two parties, such

that, it can be used for the private key algorithms. Both ends have a key pair consisting: Private Key : **d** which is less than a random number n, and

Public Key : **Q** = d \* G, where G is the Generator point of the elliptic curve.

Here, **(dA, QA)** is the Private Key - Public Key pair of A and

**(dB, QB)** be the Private Key - Public Key pair of B.

The computation goes as :

1. The end A computes K = (xK, yK) = dA \* QB

2. The end B computes L = (xL, yL) = dB \* QA

3. Since dAQB = dAdBG = dBdAG = dBQA.

4. Hence the shared secret is xK

The security in encrypting the key is that is practically impossible to find the private key dA or dB from the public key K or L.

**8.4.5 Advantage of ECC :**

ECC devices require less storage, less power, less memory, and less bandwidth than other systems. This allows you to implement cryptography in platforms that are constrained, such as wireless devices, handheld computers, smart cards, and thin-clients. It also provides a big win in situations where efficiency is important.

For example, the current key-size recommendation for legacy public schemes is 2048 bits. A vastly smaller 224-bit ECC key offers the same level of security. This advantage only increases with security level—for example, a 3072 bit legacy key and a 256 bit ECC key are equivalent—something that will be important as stronger security systems become mandated and devices get smaller.

**8.5 RSA :**

RSA is one of the earliest practicable public-key cryptography and is widely used for secure data transmission. In the cryptosystem, the encryption key is public and differs from the decryption key which is kept a secret. RSA was designed by Ron Rivest, Adi Shamir and Leonard Adleman in 1977.

**8.5.1 Operation of RSA :**

The RSA algorithm involves 3 key steps :

1) Key Generation

2) Encryption

3) Decryption

**8.5.1.1 Key Generation :**

RSA involves a Public Key and a Private Key. The public key can be known by everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers *p* and *q*.
   * For security purposes, the integers *p* and *q* should be chosen at random, and should be of similar bit-length. Prime integers can be efficiently found using a primality test.
2. Compute *n* = *p\*q*.
   * *n* is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute φ(*n*) = φ(*p*)φ(*q*) = (*p* − 1)(*q* − 1) = *n*  - (*p* + *q* -1),   
   where φ is Euler's totient function.
4. Choose an integer *e* such that 1 <  *e*  < φ(*n*) and gcd ( *e* , φ( *n* ) ) = 1;   
   i.e., *e* and φ(*n*) are coprime.
   * *e* is released as the public key exponent.
   * *e* having a short bit-length and small Hamming weight results in more efficient encryption – most commonly 216 + 1 = 65,537. However, much smaller values of *e* (such as 3) have been shown to be less secure in some settings.
5. Determine *d* as *d* ≡ *e*−1 (mod φ(*n*));  
    i.e., *d* is the multiplicative inverse of *e*(modulo φ(*n*)).

* This is more clearly stated as: solve for *d* given *d*⋅*e* ≡ 1 (mod φ(*n*))
* This is often computed using the extended Euclidean algorithm. Using the pseudocode in the *Modular integers* section, inputs *a* and *n* correspond to *e*and *φ(*n*)*, respectively.
* *d* is kept as the private key exponent.

The *public key* (n,e) consists of the modulus *n* and the public (or encryption) exponent *e*. The *private key* consists of the modulus *n* and the private (or decryption) exponent *d*, which must be kept secret. *p*, *q*, and φ(*n*) must also be kept secret because they can be used to calculate *d*.

**8.5.1.2 Encryption :**

Alice transmits her public key *(n, e)* to Bob and keeps the private key secret. Bob then wishes to send message *M*  to Alice. He first turns *M*  into an integer *m*, such that *0 ≤ m < n* by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext *c* corresponding to

419538982dbcd14d5634d61241015c92.png

This can be done quickly using the method of exponentiation by squaring. Bob then transmits *c* to Alice.

**8.5.1.3 Decryption :**

Alice can recover *m* from *c* by using her private key exponent *d* via computing

f46ed4dfc8bd4b29aa1ee699cd45724a.png

Given *m*, she can recover the original message *M* by reversing the padding scheme.

**8.5.2 Factorization Problem :**

The asymmetry is based on the practical difficulty of factoring the product of 2 large prime numbers called the Factoring problem, which is given a positive integer n, finding it's two prime factorization p and q. The best published solution to the factoring problem is the GNFS algorithm which is a subexponential time algorithm.

**8.6 JAVA SERVER PAGES :**

**8.6.1 Overview :**

**JavaServerPages** (**JSP**) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. It is used for controlling the content or appearance of Web pages through the use of Servlets, small programs that are specified in the Web page and run on the Web server to modify the Web page before it is sent to the user who requested it. To deploy and run JavaServer Pages, a compatible web server with a servlet container, such as Apache Tomcat or Jetty, is required.

Sun Microsystems, the developer of Java, also refers to the JSP technology as the Servlet application program interface (API). JSP is comparable to Microsoft's Active Server Page (ASP) technology. Whereas a Java Server Page calls a Java program that is executed by the Web server, an Active Server Page contains a script that is interpreted by a script interpreter (such as VBScript or JScript) before the page is sent to the user.

**8.6.2 Java Servlets :**

The **servlet** is a Java programming language class used to extend the capabilities of a server. Although servlets can respond to any types of requests, they are commonly used to extend the applications hosted by web servers, so they can be thought of as Java applets that run on servers instead of in web browsers. These kinds of servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET

Servlets are most often used to

* Process or store data that was submitted from an HTML form
* Provide dynamic content such as the results of a database query
* Manage state information that does not exist in the stateless HTTP protocol, such as filling the articles into the shopping cart of the appropriate customer

Servlets are a standard for implementing Java classes which respond to requests. Servlets could in principle communicate over any client–server protocol, but they are most often used with the HTTP protocol. Thus "servlet" is often used as shorthand for "HTTP servlet". Thus, a software developer may use a servlet to add dynamic content to a web server using the Java platform. To deploy and run a servlet, a web container must be used. A web container (also known as a servlet container) is essentially the component of a web server that interacts with the servlets. The web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights. Some common examples of Container Servers are :

1) Apache Tomcat

2) GlassFish

3) IBM WebSphere Application Server

4) Jetty

**8.6.3 Life Cycle of a JSP :**

The following is a typical user scenario of these methods.

1. Assume that a user requests to visit a URL.
   * The browser then generates an HTTP request for this URL.
   * This request is then sent to the appropriate server.
2. The HTTP request is received by the web server and forwarded to the servlet container.
   * The container maps this request to a particular servlet.
   * The servlet is dynamically retrieved and loaded into the address space of the container.
3. The container invokes the init() method of the servlet.
   * This method is invoked only when the servlet is first loaded into memory.
   * It is possible to pass initialization parameters to the servlet so that it may configure itself.
4. The container invokes the service() method of the servlet.
   * This method is called to process the HTTP request.
   * The servlet may read data that has been provided in the HTTP request.
   * The servlet may also formulate an HTTP response for the client.
5. The servlet remains in the container's address space and is available to process any other HTTP requests received from clients.
   * The service() method is called for each HTTP request.
6. The container may, at some point, decide to unload the servlet from its memory.
   * The algorithms by which this decision is made are specific to each container.
7. The container calls the servlet's destroy() method to relinquish any resources such as file handles that are allocated for the servlet; important data may be saved to a persistent store.
8. The memory allocated for the servlet and its objects can then be garbage collected.

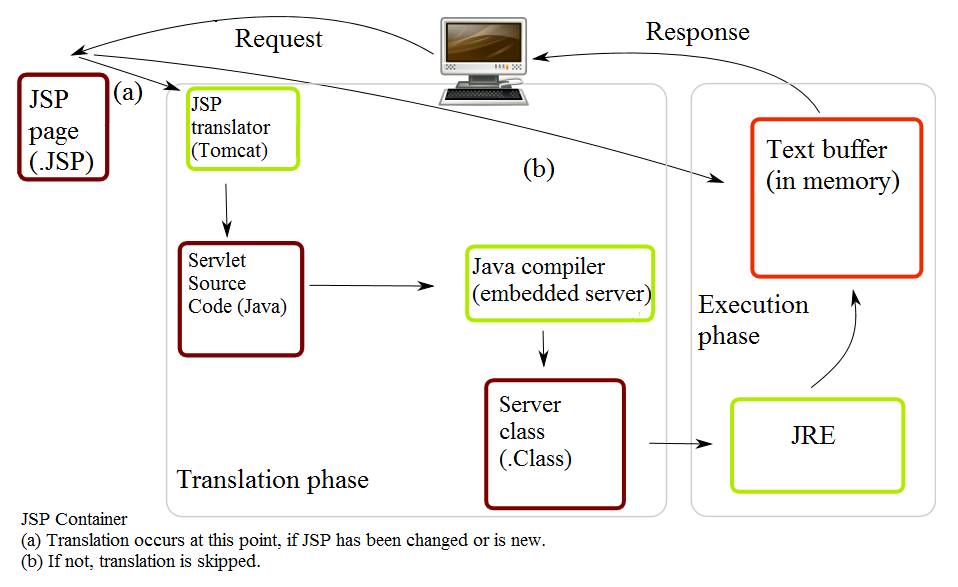
****

Fig 8.4 : Life of a JSP File

**8.6.4 Overview of JSP :**

Architecturally, JSP may be viewed as a high-level abstraction of Java servlets. JSPs are translated into servlets at runtime; each JSP servlet is cached and re-used until the original JSP is modified. JSP can be used independently or as the view component of a server-side model–view–controller design, normally with JavaBeans as the model and Java servlets (or a framework such as Apache Struts) as the controller. This is a type of Model 2 architecture.

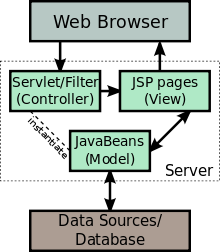


Fig 8.5 : JSP Model 2 Architecture

JSP allows Java code and certain pre-defined actions to be interleaved with static web markup content, with the resulting page being compiled and executed on the server to deliver a document. The compiled pages, as well as any dependent Java libraries, use Java bytecode rather than a native software format. Like any other Java program, they must be executed within a Java virtual machine (JVM) that integrates with the server's host operating system to provide an abstract platform-neutral environment.

JSPs are usually used to deliver HTML and XML documents, but through the use of OutputStream, they can deliver other types of data as well. The Web container creates JSP implicit objects like pageContext, servletContext, session, request & response.

**8.6.5 JSP Compiler :**

A **JavaServer Pages compiler** is a program that parses JSPs, and transforms them into executable Java Servlets. A program of this type is usually embedded into the application server and run automatically the first time a JSP is accessed, but pages may also be precompiled for better performance, or compiled as a part of the build process to test for errors.

Some JSP containers support configuring how often the container checks JSP file timestamps to see whether the page has changed. Typically, this timestamp would be set to a short interval (perhaps seconds) during software development, and a longer interval (perhaps minutes, or even never) for a deployed Web application.

****

**8.7 JAVASCRIPT :**

**8.7.1 Overview :**

**JavaScript** (**JS**) is a dynamic computer programming language. It is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed.  It is also being used in server-side programming, game development and the creation of desktop and mobile applications.

JavaScript is a prototype-based scripting language with dynamic typing and has first-class functions. Its syntax was influenced by C. JavaScript copies many names and naming conventions from Java, but the two languages are otherwise unrelated and have very different semantics. It is a multi-paradigm language, supporting object-oriented,  imperative, and functional programming styles.

The application of JavaScript in use outside of web pages—for example, in PDF documents, site-specific browsers, and desktop widgets—is also significant. Newer and faster JavaScript VMs and platforms built upon them (notablyNode.js) have also increased the popularity of JavaScript for server-side web applications. On the client side, JavaScript was traditionally implemented as an interpreted language but just-in-time compilation is now performed by recent (post-2012) browsers.

JavaScript was formalized in the ECMAScript language standard and is primarily used as part of a web browser (client-side JavaScript). This enables programmatic access to computational objects within a host environment.

**8.7.2 Use in Web Page :**

The most common use of JavaScript is to write functions that are embedded in or included from HTML pages and that interact with the Document Object Model (DOM) of the page. Some simple examples of this usage are:

* Loading new page content or submitting data to the server via AJAX without reloading the page (for example, a social network might allow the user to post status updates without leaving the page)
* Animation of page elements, fading them in and out, resizing them, moving them, etc.
* Interactive content, for example games, and playing audio and video
* Validating input values of a web form to make sure that they are acceptable before being submitted to the server.
* Transmitting information about the user's reading habits and browsing activities to various websites. Web pages frequently do this for web analytics, ad tracking, personalization or other purposes.

Because JavaScript code can run locally in a user's browser (rather than on a remote server), the browser can respond to user actions quickly, making an application more responsive. Furthermore, JavaScript code can detect user actions which HTML alone cannot, such as individual keystrokes. Applications such as Gmail take advantage of this: much of the user-interface logic is written in JavaScript, and JavaScript dispatches requests for information (such as the content of an e-mail message) to the server. The wider trend of Ajax programming similarly exploits this strength.

A JavaScript engine (also known as *JavaScript interpreter* or *JavaScript implementation*) is an interpreter that interprets JavaScript source code and executes the script accordingly. A web browser is by far the most common host environment for JavaScript. Web browsers typically create "host objects" to represent the Document Object Model (DOM) in JavaScript. The web server is another common host environment. A JavaScript web server would typically expose host objects representing HTTP request and response objects, which a JavaScript program could then interrogate and manipulate to dynamically generate web pages.

Because JavaScript is the only language that the most popular browsers share support for, it has become a target language for many frameworks in other languages, even though JavaScript was never intended to be such a language.  Despite the performance limitations inherent to its dynamic nature, the increasing speed of JavaScript engines has made the language a surprisingly feasible compilation target.

**8.7.3 Uses outside Web Pages :**

In addition to web browsers and servers, JavaScript interpreters are embedded in a number of tools. Each of these applications provides its own object model which provides access to the host environment. The core JavaScript language remains mostly the same in each application.

* Google's Chrome extensions, Opera's extensions, Apple's Safari 5 extensions, Apple's Dashboard Widgets, Microsoft's Gadgets, Yahoo! Widgets, Google Desktop Gadgets, and Serence Klipfolio are implemented using JavaScript.
* Adobe's Acrobat and Adobe Reader support JavaScript in PDF files.
* Tools in the Adobe Creative Suite, including Photoshop, Illustrator, Dreamweaver, and InDesign, allow scripting through JavaScript.
* OpenOffice.org, an office application suite, allows JavaScript to be used as a scripting language.

**8.7.4 JavaScript vs Java**

The differences between the two languages are more prominent than their similarities. Java has static typing, while JavaScript's typing is dynamic (meaning a variable can hold an object of any type and cannot be restricted). Java is loaded from compiled bytecode, while JavaScript is loaded as human-readable source code. Java's objects are class-based, while JavaScript's are prototype-based. Finally, Java did not support functional programming until Java 8, while JavaScript does, as it contains many features based on the Scheme language.

**8.8 APACHE TOMCAT :**

**Apache Tomcat**  is an open source web server and servlet container developed by the Apache Software Foundation (ASF). Tomcat implements the Java Servlet and the JavaServerPages(JSP) specifications from Sun Microsystems, and provides a "pure Java "HTTP web server environment for Javacode to run in. In the simplest config Tomcat runs in a single operating system process. The process runs a Java virtual machine (JVM). Every single HTTP request from a browser to Tomcat is processed in the Tomcat process in a separate thread. Apache Tomcat includes tools for configuration and management, but can also be configured by editing XML configuration files.



**8.9 MYSQL DATABASE :**

MySQL database is used or the login application development. The MySQL is the most used open source RDBMS for its speed, ease of use, portability, small size, performance and rich features. MySQL is a client-server application and database runs a server application. Users can access the database server through client application. The server application is known as "mysqld" and the client application is known as "mysql".

**CHAPTER 9**

**IMPLEMENTATION DETAILS**

**9.1 IMPLEMENTATION OF THE HYBRID ARCHITECTURE :**

**9.1.1 ECC - Key Pair Generation :**

The following code shows the generation of a key pair (Private Key, Public Key) using the secp192rl curve.

public class ECCKeyGeneration

{

public static void main(String[] args) throws Exception

{

KeyPairGenerator kpg;

kpg = KeyPairGenerator.getInstance("EC","SunEC");

ECGenParameterSpec ecsp;

ecsp = new ECGenParameterSpec("secp192r1");

kpg.initialize(ecsp);

KeyPair kp = kpg.genKeyPair();

PrivateKey privKey = kp.getPrivate();

PublicKey pubKey = kp.getPublic();

}

}

**9.1.2 Key Agreement :**

The following code shows the Key Agreement between two users U and V using ECDH key agreement protocol.

The Private Keys are : u and v respectively.

The Public Keys are : U = u.G and V = v.G respectively.

The SunEC implementation produces a shared value which is the first co-ordinate of the elliptic curve point computed as u.V = u.v.G = v.u.G = v.U

public class ECCKeyAgreement

{

public static void main(String[] args) throws Exception

{

KeyPairGenerator kpg;

kpg = KeyPairGenerator.getInstance("EC","SunEC");

ECGenParameterSpec ecsp;

ecsp = new ECGenParameterSpec("secp192k1");

kpg.initialize(ecsp);

//User U

KeyPair kpU = kpg.genKeyPair();

PrivateKey privKeyU = kpU.getPrivate();

PublicKey pubKeyU = kpU.getPublic();

// User V

KeyPair kpV = kpg.genKeyPair();

PrivateKey privKeyV = kpV.getPrivate();

PublicKey pubKeyV = kpV.getPublic();

KeyAgreement ecdhU = KeyAgreement.getInstance("ECDH");

ecdhU.init(privKeyU);

ecdhU.doPhase(pubKeyV,true);

KeyAgreement ecdhV = KeyAgreement.getInstance("ECDH");

ecdhV.init(privKeyV);

ecdhV.doPhase(pubKeyU,true);

}

}

**9.1.3 RSA :**

The implementation of RSA makes use of Java's BigInteger. BigInteger provides analogues to all of Java's primitive integer operators, and all relevant methods from *java.lang.Math*. Additionally, BigInteger provides operations for modular arithmetic, GCD calculation, primality testing, prime generation, bit manipulation, and a few other miscellaneous operations. This comes in handy in determining prime numbers and modInvere which is a critical part of the RSA logic.

Code snippets shows the logic of how RSA was implemented :

public RSA()

{

r = new Random();

p = BigInteger.*probablePrime*(bitlen, r);

q = BigInteger.*probablePrime*(bitlen, r);

N = p.multiply(q);

phiN = p.subtract(BigInteger.*ONE*)).multiply(q.subtract(BigInteger.*ONE*));

e = BigInteger.*probablePrime*(bitlen/2, r);

while(phiN.gcd(e).equals(BigInteger.*ONE*)==false && e.compareTo(phiN)!=1)

{

e = BigInteger.*probablePrime*(bitlen, r);

}

d = e.modInverse(phiN);

}

Code snippet for Encryption and Decryption :

Encryption Function :

public byte[] encrypt(byte[] message)

{

return (new BigInteger(message).modPow(e, N).toByteArray());

}

Decryption Function :

public byte[] decrypt(byte[] cipher)

{

return (new BigInteger(cipher).modPow(d, N).toByteArray());

}

**9.1.4 Hybrid Architecture :**

The Hybrid Architecture has been implemented by generating a Key using ECDH. This key is then used to formulate the Public and Private Key. This key pair, instead of the prime number method used to generate the Public and Private Key, is used to encrypt the plaintext and form the crypt text.

Code Snippet of embedding ECC in RSA implementation :

public HybridAlgorithm() throws Exception

{

r = new Random();

p = BigInteger.*probablePrime*(bitlen, r);

q = BigInteger.*probablePrime*(bitlen, r);

N = p.multiply(q);

phiN = (p.subtract(BigInteger.*ONE*)).multiply(q.subtract(BigInteger.*ONE*));

do{

try

{

flag=0;

e = *ECDH\_KeyGen*();

d = e.modInverse(phiN);

}

catch(ArithmeticException e)

{

System.*out*.println("No Mod Inverse Available!"+e);

flag=1;

}

}while(flag==1);

System.*out*.println("The Encryption Key Generated ( E ) : "+ e.toString(16));

System.*out*.println("The Decryption Key Generated ( D ) : "+ d.toString(16));

}

The following code snippet shows the generation of the public key (e, N) using the ECDG Key Generation. This is used to formulate the private key (d, N) by using the ModInverse Function.

**9.2 Message Digest 5 or MD5 Implementation:**

**9.2.1 Initializations :**

Code Snippet for the Initialization of :

**A) Settled Constants : ( Predefined )**

public static final int [][] *Shift* =

{ {7,12,17,22}, {5,9,14,20}, {4,11,16,23}, {6,10,15,21} };

**B) Value Declaration for A, B, C and D : ( Little Endian Format )**

public static final int *ConstA* = 0x67452301;

public static final int *ConstB* = 0xEFCDAB89;

public static final int *ConstC* = 0x98BADCFE;

public static final int *ConstD* = 0x10325476;

**C) T Value Declaration :**

private static final int[] *Tval* = new int[64];

static

{

for (int i=0; i < 64; i++)

*Tval*[i] =(int)(long)(Math.*pow*(2, 32) \* Math.*abs*(Math.*sin*(i+1)));

}

**9.2.2 Compression Function :**

**Function F :**

public static int f(int x,int y,int z)

{ return ( (x&y) | (~x&z) );

}

**Function G :**

public static int g(int x,int y,int z)

{

return ( (x&z) | (y&~z) );

}

**Function H :**

public static int h(int x,int y,int z)

{

return (x^y^z);

}

**Function I :**

public static int i(int x,int y,int z)

{

return y ^( x | ~z);

}

**9.2.3 Padding and Block Formation :**

public static String func\_md5(byte[] message)

{

int messageLenBytes = message.length; long messageLenBits = (long) messageLenBytes<<3;

int numBlocks = ((messageLenBytes+8)>>>6)+1; int totalLen = numBlocks << 6;

byte[] paddingZeroes = new byte[totalLen-messageLenBytes]; paddingZeroes[0]=(byte)0x80;

for (int i = 0; i < 8; i++)

{ paddingZeroes[paddingZeroes.length - 8 + i] = (byte)messageLenBits;

messageLenBits >>>= 8;

}

int a = *ConstA*;

int b = *ConstB*;

int c = *ConstC*;

int d = *ConstD*;

int[] buffer = new int[16];

**9.2.4 Trace of Padding :**

**PaddingZero[0] = 0x80 ->** 10000000 (8 bits)

**Message Input** -> 01100001000000000000000000000000 (32 bits)

AFTER STEP 1 (j=1,index=0)

Padding<<24

10000000+000000000000000000000000 (8+24 bits)

-> 10000000000000000000000000000000 (32 bits)

Input >> 8

01100001000000000000000000000000

-> 0000000011000010000000000000000 (32 bits (with 8 bit zeroes)

OR

10000000000000000000000000000000

(+) 00000000011000010000000000000000

10000000011000010000000000000000 (32 bits)

AFTER 2 steps (j=2,index=1)

Prev = 10000000011000010000000000000000 (PaddingZeroes = 0; So >>> 8)

New = 100000000110000100000000 (8+24 bits)

AFTER 3 steps (j=3,index=2)

Prev = 100000000110000100000000 (PaddingZeroes = 0; So >>> 8)

New = 00000000+10000000+01100001 (16+16 bits)

**BUFFER[0] - > COMPLETE**

**LENGTH PADDING J=56; index=56;**

PaddingZero[56] = MessageLenBits -> 00001000

Padding << 24 - > 00001000000000000000000000000000 (32 bits)

AFTER ROUND 1

Prev = 00001000000000000000000000000000 (PaddingZeroes = 0; So >>> 8)

New = 00000000000010000000000000000000 (8+24 bits)

AFTER ROUND 2

Prev = 00000000000010000000000000000000 (PaddingZeroes = 0; So >>> 8)

New = 00000000000000000000100000000000 (16+16 bits)

AFTER ROUND 3

Prev = 00000000000000000000100000000000 (PaddingZeroes = 0; So >>> 8)

New = 00000000000000000000000000001000 (24+8 bits)

FINAL ROUND

**9.2.5 Main MD5 Processing Block :**

for(int i=0;i<numBlocks;i++)

{

int index=i\*64;

for(int j=0;j<64;j++,index++)

buffer[j >>> 2] = ((int)((index < messageLenBytes) ? message[index] : paddingZeroes[index - messageLenBytes]) << 24) | (buffer[j >>> 2] >>> 8);

int h0 = a;

int h1 = b;

int h2 = c;

int h3 = d;

for(int j=0,k=0;j<64;j++,k++)

{

int block = j/16;

int f=0;

int g = j;

switch(block)

{

case 0:

f = *f*(b,c,d);

break;

case 1:

f = *g*(b,c,d);

g = (1 + 5\*j)%16;

break;

case 2:

f = *h*(b,c,d);

g = (5+3\*j)%16;

break;

case 3:

f = *i*(b,c,d);

g = (7\*j)%16;

break;

}

int temp = b + Integer.*rotateLeft*(a + f + buffer[g] + *Tval*[j], *Shift*[block][k%4]);

a=d;

d=c;

c=b;

b=temp;

}

a = a + h0;

b = b + h1;

c = c + h2;

d = d + h3;

}

**9.3 CLIENT-SERVER LOGIN :**

**9.3.1 Client Login Form :**

The client login form has been implemented using JavaServerPages. The implementation is a simple client-server login where the user enters his login credentials in the Login Form. The password is thereby encrypted on the client side using Client Side Encryption using the Hybrid algorithm and then transmitted across the transmission side. A user can enter his/her userID and password. If he is already registered then the details are backed up from the database and will be matched with the entered credentials. If they are matched successfully then a success page will be displayed otherwise an error page will be displayed.

Code Snippet of the Login Form JSP :

<body>

<center>

<form action=*LoginServlet*>

<table border=*1*>

<tr>

<th colspan=*"2"*>Login Form</th>

</tr>

<tr>

<td>Enter your UserId:</td>

<td><input type=*text* name=*userId*></td>

</tr>

<tr>

<td>Enter your Password:</td>

<td><input type=*password* name=*pwd*></td>

</tr>

<tr>

<td><input type=*submit* value=*submit*></td>

<td><input type=*reset* value=*refresh*></td>

</tr>

</table>

</form>

</center>

</body>

Code Snippet for Login Success :

<body>

<center>

<center>Login Success</center>Welcome <%= session.getAttribute("FirstName") + " " + session.getAttribute("LastName") %>

</center>

</body>

Code Snippet for Login Failure :

<body>

<center>

<center>Login Failed</center>

Incorrect User ID / Password. Please try again! <br>

<center><input type=*button* name=*back* value=*Back* onClick="history.go(- 1);" /></center>

</center>

</body>

**9.3.2 MySQL Database :**

Starting the MySQL Server :

1) Start --- > run and Open the Command Propmt.

2) Go to the bin directory of the "basedir" path.

3) Run the command "mysqld --console"

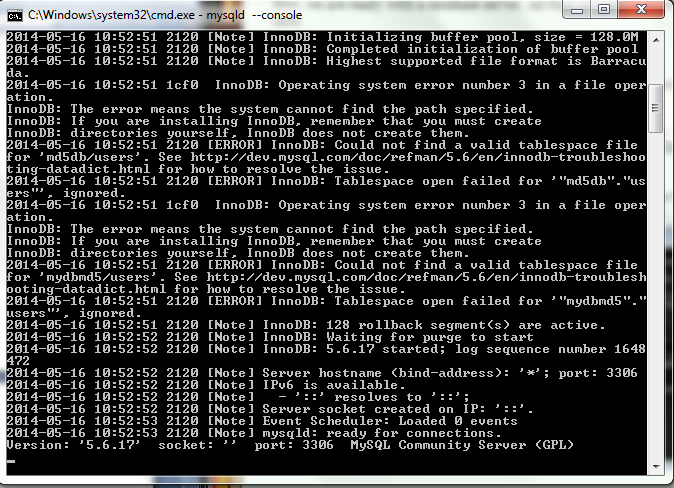


Fig 9.1 : MySQL Server End Configuration

Starting the MySQL Client :

1) Start --- > run and Open the Command Propmt.

2) Go to the bin directory of the "basedir" path.

3) Run the command "mysql -u root"

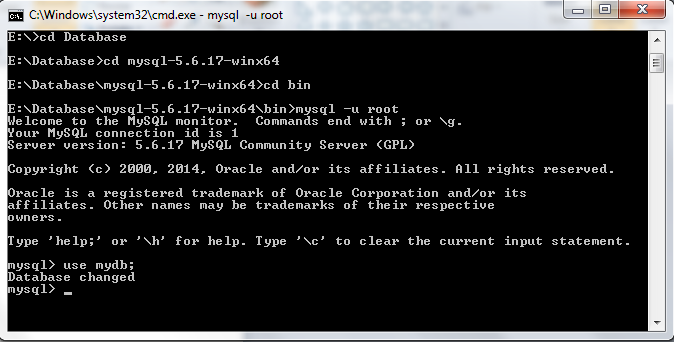


Fig 9.2 : MySQL Client End Configuration

The Database Entries :

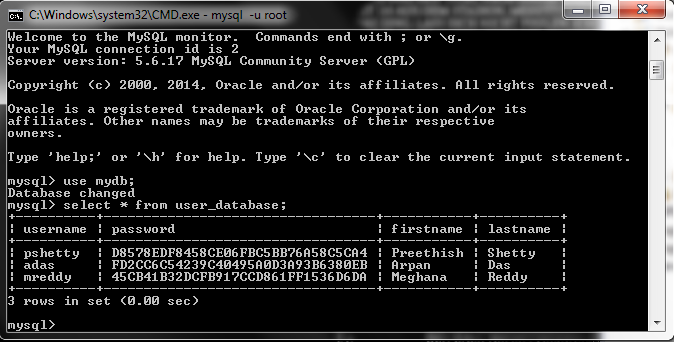


Fig 9.3 : MySQL Database Entries

The following screenshot shows the entries made in the MySQL Database. The crucial point here is that unlike other systems the passwords are stored in hashed format rather than in plaintext. This makes it harder for attackers to retrieve passwords if the server database has been compromised.

**CHAPTER 10**

**EXPERIMENTAL RESULTS AND**

**ANALYSIS OF THE RESULT**

**10.1 RESULTS OF DIFFERENT ALGORITHM IMPLEMENTATION :**

**10.1.1 MD5 :**

8th Semester Project -> Hybrid Cryptography Algorithm : Part 1 "MD5 Encryption"!

Final Result :

0x0CC175B9C0F1B6A831C399E269772661 <= "a"

0x900150983CD24FB0D6963F7D28E17F72 <= "abc"

0xC3FCD3D76192E4007DFB496CCA67E13B <= "abcdefghijklmnopqrstuvwxyz"

0x781E5E245D69B566979B86E28D23F2C7 <= "0123456789"

0x605EEE1A58A4E6976FB9AA8702851043 <= "$123#"

0x0ECCE0FEB23F019DA9FF972B0D3DCF8B <= "Preethish Shetty"

# Note : The first 5 inputs are the standard inputs given as documented in the RFC 1321

The following result shows us that irrespective of the input given ( in terms of length or format ) the resulting output is a 128 bit hashed value.

**10.1.2 ECDH Key Agreement:**

8th Semester Project -> Hybrid Cryptography Algorithm : Part 2 "ECDH"!

User U : Sun EC private key, 192 bits

private value: 5312906147488083186849133445977397170761900026882457592533

parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)

User U : Sun EC public key, 192 bits

public x coord: 2538259584277954655595283899803824248712800813402319133790

public y coord: 6130640835612471165297700540218227170252256006317855068119

parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)

User V : Sun EC private key, 192 bits

private value: 5467593055343945690207326757869365642690847375022661843269

parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)

User V : Sun EC public key, 192 bits

public x coord: 5382616536622678511250638513997978136810236174183458898826

public y coord: 5664305566821132000110814688970235716855107996945454526306

parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)

Secret computed by U : 0x48C87544BD9CB1FE9A0F4B330CB66E99DB3433C39593F4F2

Secret computed by V : 0x48C87544BD9CB1FE9A0F4B330CB66E99DB3433C39593F4F2

The following result shows in detail the various parameters generated by the ECDH Implementation. The output statements reflect the Private Key value and the parameter used to generate the given value namely secp192r1. In addition, it prints out the x and y co-ordinate that satisfies the curve which forms the Public Key.

**10.1.3 RSA :**

8th Semester Project -> Hybrid Cryptography Algorithm : Part 3 "RSA"!

The E value is : c9d794b77eae0a74c4123034fd03f282b7bd94c191a9e2b6c702756abd96875f24823da2c44c2c6667f643c3d9e617eac095e50c18852416bab03439eacd03d1

The D value is : 

Enter the input string :

Preethish Shetty

The Input : Preethish Shetty

The Input (In Bytes) : 801141011011161041051151043283104101116116121

The Cipher : 8-138624-15-83-71104721081074551-67483-126-97452987-13-99-64-93-897038-297188-112-9189-31-4277-11287-24-29899-524106-15112-8418-4912354-127-67-662743707259725982-104112-99127-64-17-113-251113-96-7966-92-12311963478601288-17-89-7611727-522648114-30-60-64107819138914-4741111-117773372885711590-77-125108-70396997-109-474710167105125-109-96-64-1-78-10010030-5187-56-3743-6089039-31-12062-95-10-2910234-9423630113-21-4-106111-71-779371-82-6910372-26-1051757116-74-49299830115-104-69-57-6777-24-17-9-1-5330-12241-42-25102-7686-28-4-8590-5489-10463-64-53-1837995718811566-712036621-125-66-991-12-23-102-4968-7611912774-114-844621-3-183-27489081-506950-101-119-92

The Message (In Bytes) : 801141011011161041051151043283104101116116121

The Message : Preethish Shetty

The following output traces the encryption and decryption of a given random input using the RSA Algorithm implemented. It prints the values of the encryption and decryption key. In addition, it prints the Cipher Text got after encrypting the plaintext using the RSA algorithm. Thereafter, the plaintext is printed after decrypting the given crypt text.

**10.1.4 Hybrid Architecture :**

8th Semester Project -> Hybrid Cryptography Algorithm :

User U : Sun EC private key, 192 bits

private value: 2870752083430654097378828650265401216106987568972883152226

parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)

User U : Sun EC public key, 192 bits

public x coord: 1205942086840892257962867892867715193802868873397068333831

public y coord: 4525235195180279630920169080866952598043303934441269822290

parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)

User V : Sun EC private key, 192 bits

private value: 856578135014193782191041285564750306542917942368137383423

parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)

User V : Sun EC public key, 192 bits

public x coord: 1296380355831930334790492564458586639474778368379398569316

public y coord: 4580076179085672497529406893857754519031545359490647841344

parameters: secp192r1 [NIST P-192, X9.62 prime192v1] (1.2.840.10045.3.1.1)

Secret computed by U : 0x5A3B646CDD5EBC73E267FAF0905A29B59BAD5A5B1464694D

Secret computed by V : 0x5A3B646CDD5EBC73E267FAF0905A29B59BAD5A5B1464694D

Secret computed by User : 5A3B646CDD5EBC73E267FAF0905A29B59BAD5A5B1464694D

The Key Generated by ECDH ( E ) : 5a3b646cdd5ebc73e267faf0905a29b59bad5a5b1464694d

The Key Generated by ECDH ( D ) : 

Enter the input string :

Preethish Shetty!!

The input string : Preethish Shetty!!

The Input (in bytes) : 8011410110111610410511510432831041011161161213333

The Cipher : 37-7869-3841-88-12810886-53949-28100-67-93-126-59-10086-102-11139-8811837-97709511237-3742-19-10-77557806151-1496-68559575-44-63-4346-121-75524886-25-48709-8841-287154112-107112-4559-17-82-123104-115-58312633-962550124-109118-5615-51-9100108112-187873-12-3861127-2031-25121112796377100-43-45-74-1069-37-52-774017-91-75392415-76-61-6937-103-118567126-16112-98-76-327993-111-125-99-70-3748114-5165-103109-102-381107-74-1126936-8834-5880-4913-7317-126-2310884-8-5-7838-106-43-5776-11112556-5-10466-585-125-7815-81-102-125-44-110-98-132959-1933564411-61-548852-311954-83126-9612-89-1-103625612633-25-67-83-107-121-93865457-888-61-128-8178-6-210951-15-33-39-22-124-8897-12-7310780-985089

The Message (In Bytes) : 8011410110111610410511510432831041011161161213333

The Message : Preethish Shetty!!

The following output traces the encryption and decryption of a given random input using the Hybrid Architecture Implemented. It prints the values of the encryption and decryption key which is received by embedding the value generated by the ECDH function in the program. In addition, it prints the Cipher Text got after encrypting the plaintext using the RSA algorithm. Thereafter, the plaintext is printed after decrypting the given crypt text.

**10.2 CLIENT - SERVER LOGIN :**

**10.2.1 Client Login Form :**

The following screenshot shows the Login Form where the user is asked to enter his User Credentials. On entering the UserID and Password he/she has to click on the Submit button to be redirected to the appropriate page.

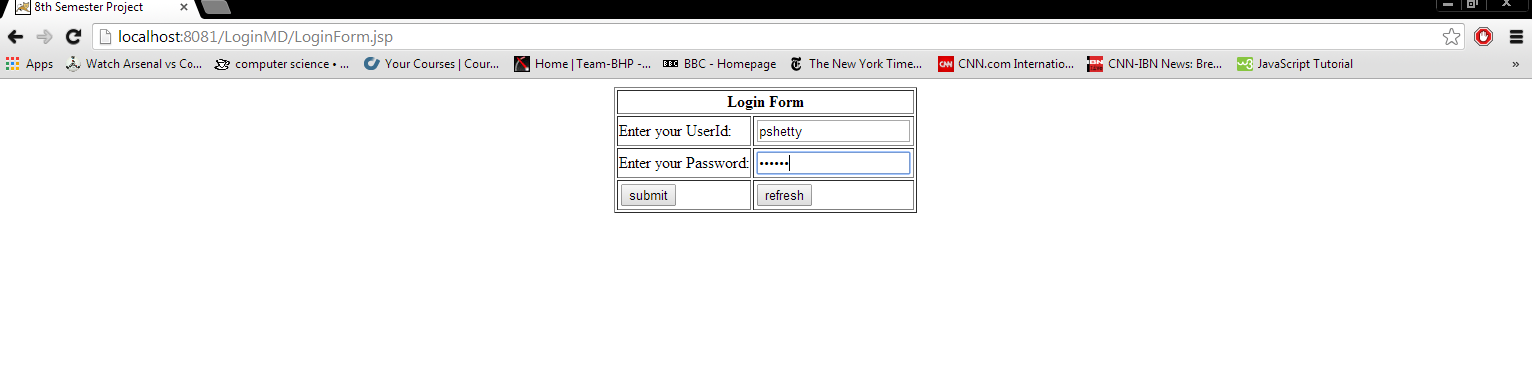


Fig 10.1 : User Login Page

**10.2.2 Client Login Success Page :**

When the User enters a credential that matches an entry in the MySQL database, he/she is redirected to the Login Success Page where the User First Name and Second Name is embedded into a generic Welcome statement.

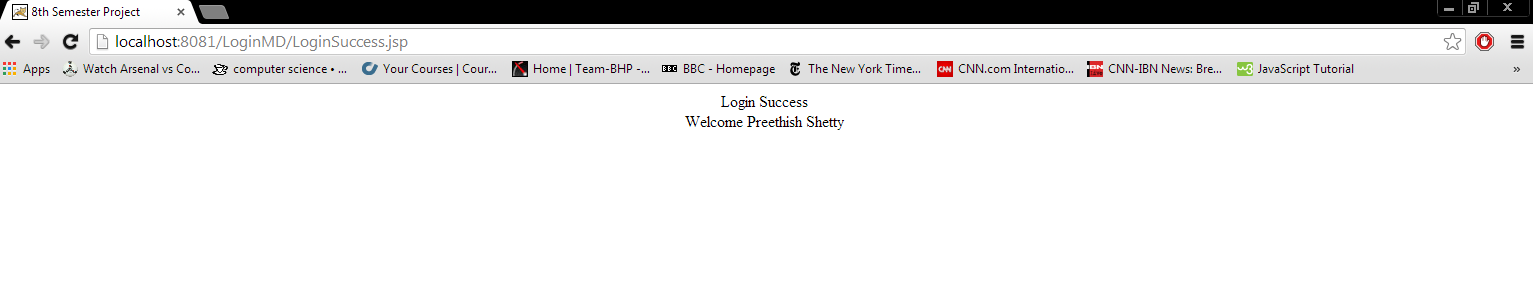
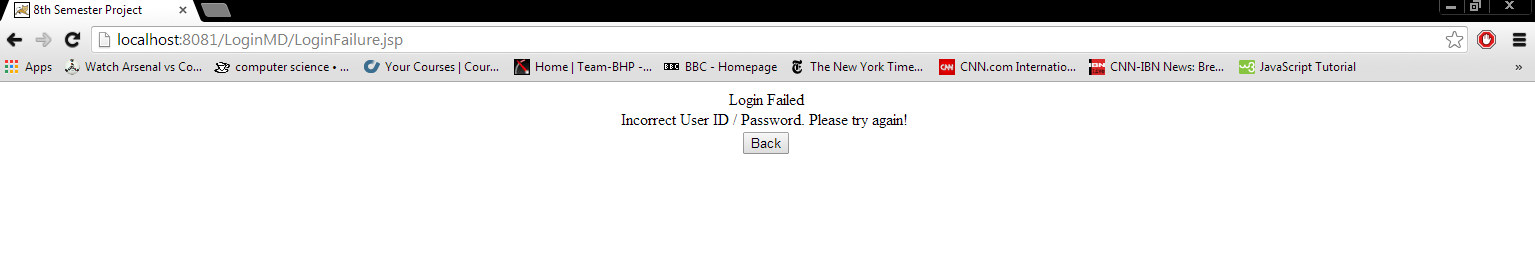
****

Fig 10.2 : Successful Login Page

**10.2.3 Client Login Failure Page :**

When the User enters a credential that does not match an entry in the MySQL database, he/she is redirected to the Login Failure Page where a Login Failure statement is presented and an option to be redirected to the Login Page is present.

****Fig 10.3 : Failure of Login Page

**Chapter 11**

**CONCLUSION**

According to a report by CNNMoney dated December4, 2013, hackers have stolen usernames and passwords for nearly 2 million accounts of Facebook, Google, Twitter, Yahoo and others. With us approaching the digital era where everything from online shopping to all sorts of media transfer occurs; it is pivotal that the user credential is safe and secure. Potential loss of passwords can leads to losses of intellectual property, sensitive information and funds in more severe and dire cases. It is of extreme importance to get stringent password protection methods in force.

With this project an attempt was made to strengthen traditional User Logins. The project aimed at implementing a two-fold security measure. First, the project aimed at ensuring client side encryption which prevents passwords being transmitted to the server side in plaintext. This ensures that attackers monitoring the transmission medium are not able to intercept the password and use it for malicious purposes. Secondly, the project highlighted the need for a stronger and more robust algorithm. Current encryption standards have been attacked and hence proven no longer safe as an encryption standard. The concept of a hybrid encryption algorithm is one that is more stronger and robust, and less prone to attacks. In addition, This hybrid ensures a more robust and lightweight encryption algorithm that provides integrity, authentication, confidentiality and reliability.

**CHAPTER 12**

**REFERENCES**

1. **Xialing Zheng, Jidong Jin** (2012), *Research for the Application and Safety of MD5 Algorithm in Password Authentication*
2. **Manali J Dubal, Mahesh T R** and **Pinaki A Ghosh** (2011), *Design of a new security algorithm using hyrbid cryptography architecture.*
3. **V. Gayoso Martinez** and **L Hernandez Encinas** (2013), *Implementing ECC with Java Standard Edition 7*
4. **Wang Yu, Chen Jianhua, He Debiao** (2009), *A new collision attack on MD5*
5. **William Stallings,** *Cryptography and Network Security - Principles and Practices* ( Third Edition )
6. **Dario Forte,** (2009), *The Death of MD5*
7. **RFC 1321, R. Rivest** (1992) - The MD5 Message-Digest Algorithm

**PROJECT DETAILS**

|  |  |  |  |
| --- | --- | --- | --- |
| *Student Details* | | | |
| **Student Name** | | Preethish Shetty | |
| Register Number | | 100905013 | |
| Section | | A | |
| Roll number | | 08 | |
| Email Address | | shetty.preethish@gmail.com | |
| Phone Number (M) | | +91-9900405580 | |
| *Project Details* | | | |
| **Project Title** | | Implementation of a new security algorithm using hybrid cryptography for password protection | |
| Project Duration | | 4 months | |
| Date of Reporting | | 13-01-2014 | |
| *Organization Details* | | | |
| **Organization Name** | | | Manipal Institute of Technology, Manipal |
| Full postal address with pin code | | | Manipal Institute of Technology,  Manipal – 576 104 (Karnataka State),  INDIA |
| Website address | | | http://www.manipal.edu/institutions/engineeringandit/mit/pages/welcome.aspx |
| *Internal Guide Details* | | | |
| **Faculty Name** | Dr. Krishnamoorthi Makkithaya | | |
| Full contact address with pin code | | Dept of Computer Science and Engg,  Manipal Institute of Technology,  Manipal – 576 104 (Karnataka State),  INDIA | |
| Email address | | k.moorthi@manipal.edu | |