Secure User Authentication for Corporate Sector

Jitendra Singh and R. P. Mahapatra

Abstract-**The aim of this article is to describes and develop a model that works on the issues related to setting up Secure User Authentication for Corporate Sector Systems. The range of front-office applications in corporate sector sometimes makes it difficult or even impossible to exchange information between services. Each service has its own applications. The information in these applications either only concerns the service itself, or must be shared with other services treating the same customer. In addition, from the point of view of the central information system, some of these specific applications are black boxes that manage themselves (but unfortunately do not correct themselves). The isolation of the applications affects customer management on inter-service interactions. Thus, inter-service requests use paper forms. The client details have been designed to meet this requirement of controlled sharing of data. It is a central application containing the account and personal information needed to coordinate the client's requirements like bill payment, reports, monthly statements, etc. It usually contains a summary of the data, and may also be coupled to a server-providing full. Unfortunately, this details does not completely solve all problems linked to access to a customer's data. They should get a single authentication module. In other words, once they should automatically be logged on to the applications. It is quite tiring for them to have to remember the separate user ids and passwords and use them during the application logons. The corporate sector can establish and maintain an effective SSO environment that maximizes security and facilitates compliance. The method or the protocol we had used is Kerberos As the result we have provided a comprehensive and integrated solution that can helps corporate sector by simplify integration between systems and applications across the enterprise and with other organizations across the Internet, facilitate security and compliance initiatives by helping to reduce exposure and providing audit and tracking reports. Enhance end-user experience through single sign-on and seamless access to multiple services. Create revenue-generating opportunities by enabling controlled access to customer information and supporting partnerships with financial service providers. Reduce administration overhead, including help-desk costs, by providing a single authentication module.**

I. INTRODUCTION

Many real life systems use an authentication protocol called Kerberos [5]. The basic for Kerberos is another protocol called Needham-Shroeder. Designed at MIT to allow the workstation to allow network resources in the secure manner.

Jitendra Singh, Sr. Lecturer, CS Dept. SRM-IMT, Modinagar Campus, SRM University, Chennai. Email: jitendra.jit@gmail.com

R. P. Mahapatra, Assistant Professor, CSE Dept. SRM-IMT, Modinagar Campus, SRM University, Chennai . Email: mahapatra.rp@gmail.com

The name Kerberos signifies a multi-headed dog in the Greek mythology [5] (apparently used to keep outsiders away) Version 4 of Kerberos is found in most practical implementation. However, Version 5 is also in use now[6].
How does Kerberos work [1]?
There are four parties involve in Kerberos protocol:

1. Alice: the client workstation.
2. Authentication server (AS): verifies (authentication) the user during login.
3. Ticket granting server (TGS): issues ticket to certify proof of identification.
4. Bob: the server offering service such as network printing, file sharing or an application program.

The job of AS is to authentication every use at the login time. AS shares unique secure password with every user. The job of TGS is to certify to the server in the network that user is really what the claim to be. For proving this, the mechanism of tickets (while allow entry into a server, just as a ticket which allow entry into a server, just as a ticket allow parking a car are entering a music concert is used.
There are three primary steps in the Kerberos protocol.
Step1: login
To start with Alice, the user, site down at an arbitrary publication workstation and enter her name .the workstation sends her name in plain text to the AS, as shown figure 1. In response, the AS performs several actions. It first creates a package of the user name (Alice) and randomly generated session key (KS). It encrypts this package with symmetric key that the AS shares with the ticket-granting server (TGS). The output of this step is called as the ticket granting ticket (TGT). Note that TGT can be opened only by the TGT since only it possesses the corresponding symmetric key for decryption.

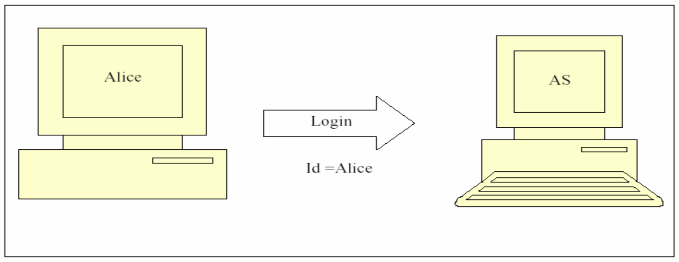

Figure 1: Alice sends a login request to AS

The AS then combine the TGT with the session key (KS) and encrypt the two together using a symmetric key derived from the password of Alice (KA). Note that the final out put can therefore, be opened only by Alice. This is shown in figure 2.

After this message is Received, Alice's workstation asks her for the password. When Alice enters it .the workstation generated the symmetric key (KA) derived from the password (in the same manner as AS would have done earlier) and uses that key to extract the session key (KS) and the ticket granting ticket (TGT).

The workstation destroys the password of Alice from its memory immediately; to prevent the attacker from stealing it, note that Alice cannot open the TGT, as it is encrypted [3] with the key of TGS.

Step 2: Obtaining the Service Granting Ticket (SGT)
Now let us assume that after a successful login, Alice want make use of Bob-the email server for some email communication for this Alice would inform her workstation that she need to contact Bob. Therefore, Alice need to ticket to communicate with Bob, at this juncture, Alice workstation create s a message intended for the ticket granting server (TGS). Which contain the following item:

1. The TGT as in one step 1
2. The id of the server (Bob) whose service Alice is interested in, the current time stamp, encrypted with the same session key (KS). This is shown in figure 3.

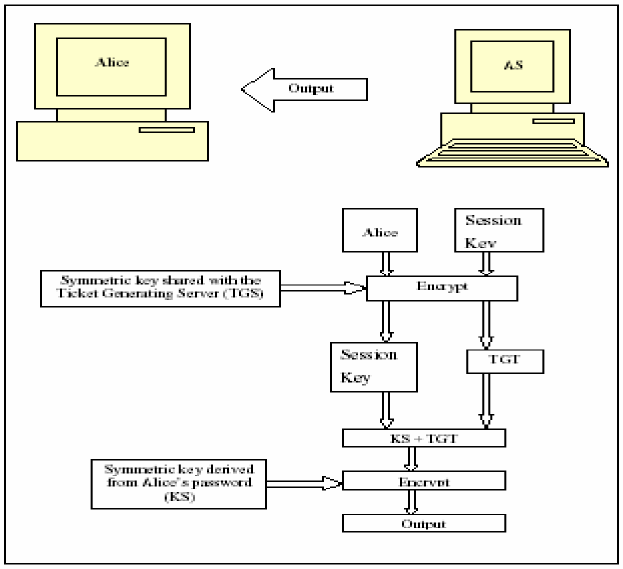
Alice
Output
AS
Alice
Session Key
Symmetric key shared with the Ticket Generating Server (TGS)
Encrypt
Session Key
TGT
KS + TGT
Symmetric key derived from Alice's password (KS)
Encrypt
Output

Figure 2: AS sends back-encrypted session key and TGT to Alice

As we know, the TGT is encrypted with the secret key of the ticket-granting server (TGS). Therefore, only the TGS can open it. This also serves as proof to the TGS that the message indeed came from Alice .why? This is because, if you remember, the TGT was created by the AS (remember that only AS and TGT know the secret key of TGS) further more, the TGT and the KS were encrypted together by the AS with secret key derived from the password of Alice[2]. Therefore, only Alice could have opened that package and retrieved the TGT.

Once the TGS is satisfied of the credentials of Alice the TGS creates a session key KAB, for Alice to have secure commutation with Bob. TGS send it twice to Alice :once combined with Bob's id (Alice) and encrypted with the session key (KS), and second time, combine with Alice's id (Alice) and encrypted with Bob's secret key (KB). This is shown in figure 4
Note that attacker Tom can try and obtain the first message in this step sent by Alice, and attempt a replay attack. However, this would fail as the message from Alice contains the encrypted timestamp. Tom cannot replace the timestamp, because he does not have the session key (KS). Even the tom attempts a reply attack really quickly, all that he will get back is the above message from TGS, which tom can not open, as he does not have access to either Bob's secret key or the session key (KS).

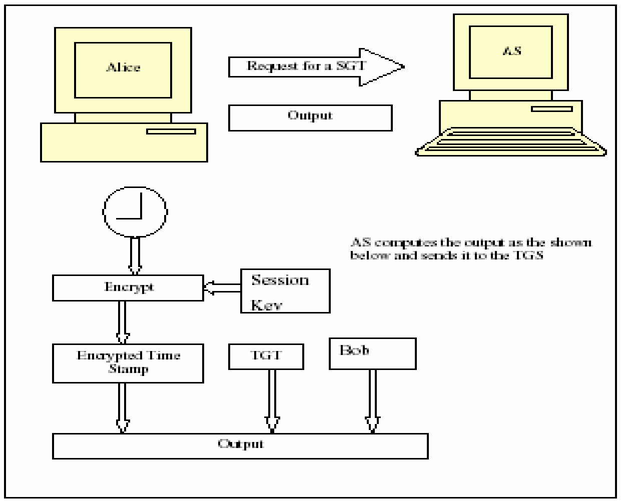

Figure 3: Alice sends a request for a SGT to this TGS

Step 3: User contacts Bob for accessing the server
Alice can now send KAB to Bob in order to enter a session with him. Since this exchange is also desired to be secure, Alice can simply forward KAB encrypted with bobs secret key (which she had also received from the TGS in the previous step) to Bob. This will ensure that only Bob can access KAB. Furthermore, to guard against reply attack, Alice also sends the time stamp, encrypted with KAB to Bob. This is shown in fig 5.
Since only Bob has his secret, he uses it to first obtain the information (Alice + KAB) from this, it gets the key KAB which he uses to decrypt the encrypt time stamp value.
Now how would Alice know if Bob received KAB correctly or not? In order to satisfy this query, Bob now adds 1 to the timestamp send by Alice, encrypt the result with KAB and send it back to Alice. This is shown in fig 6. Since only Alice, and Bob know KAB, Alice can open this packet, and verify the timestamp incremented by Bob was indeed the one sent by her to Bob in the first place.
Now Alice and Bob can communicate securely with each other. They would use the shared secret key KAB to encrypt massage before sending and also to encrypt the encrypted message received from each other.

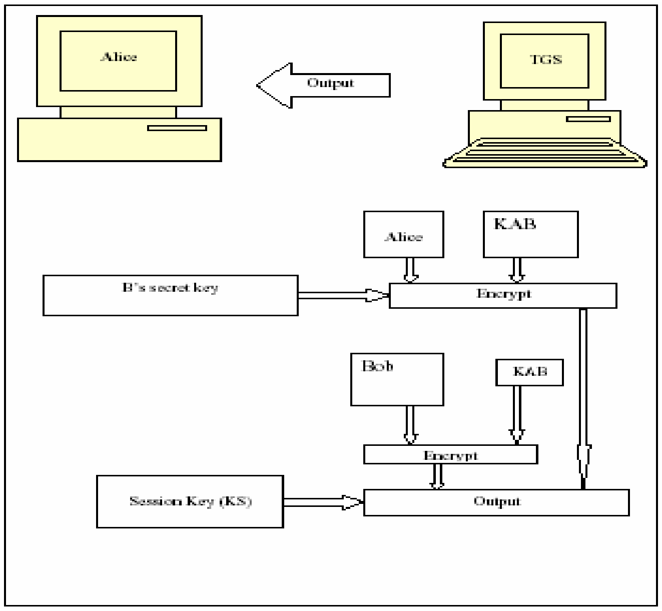

Figure 4: TGT sends responds to Alice

An interesting point here is that if Alice now want to communication with another server any carol, she simply need to obtain another share key from the TGS only now specifying carol instead ob Bob in her message .the TGS will do needful, as explained earlier. The out come is that Alice can now access all the resources of network in similar manner, each time obtaining a unique ticket (secret key) from the TGS to communicate with a different resource .of course, if Alice want to continue communicating with Bob alone, she need not obtain a new ticket every time. Only for the first time that she want to communicate with a server that she need to contact TGS and obtain a ticket. Also, Alice's password never leaves her workstation, adding to the security.
.

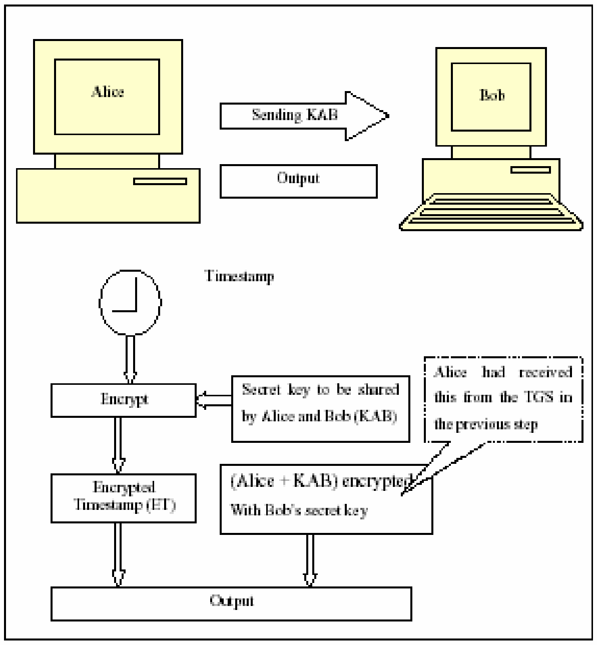
Alice
Sending KAB
Bob
Output
Timestamp
Encrypt
Secret key to be shared by Alice and Bob (KAB)
Alice had received this from the TGS in the previous step
Encrypted Timestamp (ET)
(Alice + KAB) encrypted With Bob's secret key
Output

Figure 5: Alice sends KAB securely to Bob

Since Alice needs to authenticate or sign on only once, this mechanism is called a single sign on (SSO). Alice need not prove her identity to every resource in the network individually. She needs to authenticate herself only to the central AS only once. That is good enough for the all the other servers/network resource to be convinced of Alice's identity

SSO is a very important concept for cooperate network, because they grow over period of time with multiple authentication mechanism and divers implementation.
These can be segregated into single, uniform authentication mechanism using SSO. In fact, Microsoft passport technology on the Internet is also based on this philosophy. Microsoft window NT also uses the Kerberos mechanism heavily. This is also why once you log on to a windows NT workstation, you can access your email and other secret resources without requiring explicit logons, as long as the correct mapping s are done by the system administrator. Clearly, not every server in the world would trust a single AS and TGS. Therefore, the designers of Kerberos provide a support for multiple realms, each having its own AS and TGS[8].

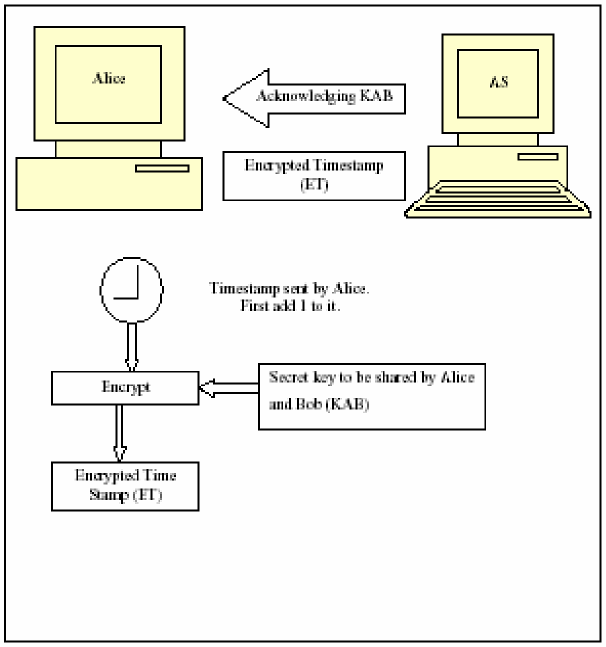
Alice
Acknowledging KAB
AS
Encrypted Timestamp (ET)
Timestamp sent by Alice. First add 1 to it.
Encrypt
Secret key to be shared by Alice and Bob (KAB)
Encrypted Time Stamp (ET)

Figure 6: Bob acknowledges the receipt of KAB

Version 5 of Kerberos overcome some of the shortcoming of version 4.version 4 demands the use of DES. Version 5 allows flexibility in the term of allowing the choice of other algorithms. Version 4 depends on IP address as identifier. However, version 5 allow the use of other type as well (for this, it tags network address with type and length)

II. SINGLE SIGN ON (SSO) APPROACHES

Single sign on (SSO)[4] solutions are based on one of the two broad levels approaches: the script approach, and the agent approach.

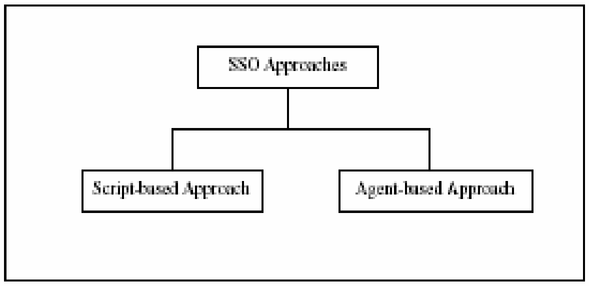

Figure 7: Approaches

Scripting:

In the scripting technique, the SSO software mimics user actions. It does this by interpreting a program, which simulates the user depressing keyboard keys, and reacting to individual end–system sign-on prompts. The SSO product itself holds and manages the different sets of authentication information from its databases and inserts it in the data stream simulated to be from the user to enter information at the appropriate points. If needed, the script can be programmed to prompt the user to enter information at appropriates places in the script.

In this approach, batch files and scripts containing authentication information (usually user id and passwords along with the login command s, if any) for each application/platform are created. When a user requests an access, a script runs in the background, and performs the same commands/tasks that the user performs. The scripts can contain macros to replay user keystrokes/commands within a shell. This is easy for users, but quite tough for system administrators, as they have to first play a role in the creation of scripts have to maintain these script securely (as they contain user ids and passwords, etc.) and have to also ensure change coordination when users wants to change their passwords [1].

Agents:

In the agent-based approach, every Web server running an application must have a piece of software, called as an agent. Additionally, there is a single SSO server, which interacts with the user database to validate user credentials. Agents interact with SSO server to achieve single sign on.

Whenever a user wants to access an application /a site participating in SSO, the agent sitting on the particular Web server intercepts the user's HTTP request and checks for the presence of a cookie. There are two possibilities now:

1. If the cookie is not present, the agent sends the login page to the user, where the user must enter the SSO user id and password. The login request goes to the SSO server, which validates the user credentials, and if this process is successful, it creates a cookie for the user.

2. If the cookie is present, the agent opens it, validates its contents, and it they found ok, and allows further processing of user's request. [1]

III. IMPLEMENTATION

Single sign on (SSO) solutions are based on one of the two broad levels approaches: the script approach, and the agent approach. We can choose either one. However. Since the agent approach is considered more suitable for web-based application. We shall use it here, as we know an agent is small program that runs on each of web servers that host an application within application framework. This agent helps coordinate the SSO workflow in term of user authentication and session handling.

The corporate sector's applications run on the Intel-based servers on Windows NT 4.0 operating system. These applications are developed by using JSP/ASP and SQL server 6.0.the web server is Tomcat server/Microsoft 's Internet information system (IIS) 4.0. There is an involvement of Microsoft transaction server (MTS) for transaction handling. However the SSO requirement need not be concerned with it.

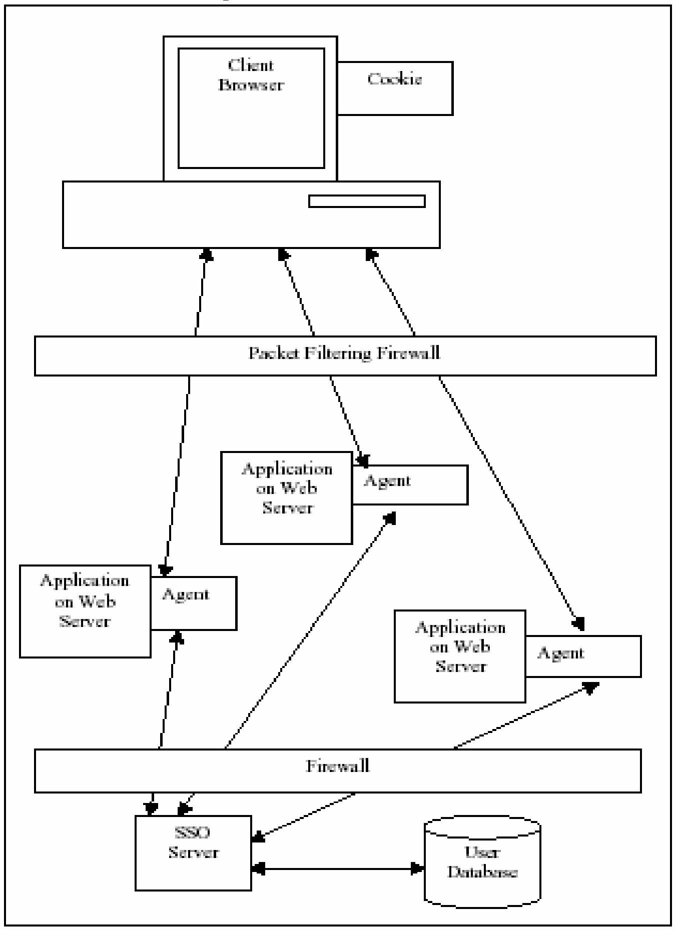
Client Browser
Cookie
Packet Filtering Firewall
Application on Web Server
Agent
Application on Web Server
Agent
Application on Web Server
Agent
Firewall
SSO Server
User Database

Figure 8: SSO Architecture

In order to develop the agent no special hardware or software requirement are visualized. The agent is simple program sitting on the IIS web server, and they can be written in the form of ISAPI application (i.e. the filter on the IIS web server). The broad level solution architecture is depicted in figure 8.

As we can see the SSO architecture [1] contains two main pieces: the agent sitting on web server, and dedicated SSO server, the purpose of these two pieces is as follow:

1. Agent sitting on web server: An agent would intercept every HTTP request arriving at the web server. There is one agent per web server, which hosts an application. It interacts with the client browser on the user side and with the SSO server on the application side.

Dedicated SSO server: The SSO server user transient cookies to provide session management functionalities. A cookie contains information such as the user id, session id, session creation time, session expiration time, etc.
The application flow would be as follow.
1. For every HTTP request that is intercepted, the agent will look for the existence of valid cookies. There are two possibilities:

I) If the cookie is not found, it will initiate a challenges screen to allow the user to enter her credentials. The credentials may be simple user id/password, or user id and digital certificate, depending on the mechanism chosen for user authentication .the agent would receive these details enter by the user, and forward them to the SSO server, which would validate them against the user data base .if the user is authenticated successfully, the SSO server will respond back with a credential token. The agent may forward part of the token to the client browser as cookies. The cookies may contain basic information like session identifier, session expiry time, etc.

II) If the agent finds an exiting cookie along with an intercepted HTTP request, it will request the SSO server to decrypt the same and determine whether:

- The user is already authenticated.
- The authentication is still valid.
- The user can access the application associated with this agent

If the authentication has expired, it will ask the user to provide authentication details once again.

2. The SSO server will receive authentication request from the agent .it will then initiate a call to an authentication JSP. This JSP will authentication the user against the user database, and returns success or failure.

On successful authentication, the SSO server will build a credential token with some information and return the whole or part of this token to the agent.

If the user already authenticated and the agent request for verification, the SSO server will determine whether the user is allowed an access to the system. Accordingly, it will initiated the authentication process or will inform the agent to allow user to access the application, if the session is still valid.

IV. CONCLUSION

The corporate sector can establish and maintain an effective SSO environment that maximizes security and facilitates compliance. We provide a comprehensive and integrated solution that helps corporate sector:

1. Provides integration with other authentication (like smart cards) Secure the primary network logon and access to application credentials and other single sign-on data with strong two-factor authentication.
2. Reduces help desk support costs by up to 40%*With Secure SSO, users no longer need to remember as many credential sets with single sign on, meaning fewer calls to reset passwords or unlock accounts.
3. Enhances user convenience and increases productivity Users access multiple systems and applications using a single secure login. Reduce employee time spent logging on and waiting for resolution from help desk password resets.
4. Reduces exposure to risk Eliminate the temptation for users to write down passwords or create simple, guessable static credentials. Create strong password policies that don’t require less secure password sharing or synchronization methods.
5. Assists with compliance Secure SSO can assist with compliance of mandates and government regulations for identity, privacy, policy enforcement, and audit and authentication services.

V. RECOMMENDATION AND FUTURE WORK

The major direction is to adapt these ideas to a file service that supports a more secure inter-branch payment and the technology to achieve non-repudiation with more enhancements.

VI. ACKNOWLEDGEMENT

There are far too many people to try to thank them all; many people have contributed to the development of this paper. This is only a partial listing....

1. I also express my gratitude to Dr. S. P. Sabarwal, Director, SRM-Institute of Management & Technology, Modinagar and all the faculty members for providing me invaluable support, guidance, help and inspiration all through project.

2. I owe my deep regards and honour to Dr. K. K. Bhardwaj, Professor, School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi for his unconditional encouragement, motivation and guidance throughout the entire project.

VII. REFERENCES

[1] Kahate, Network Security and Cryptography, Tata McGraw Hill.
[2] Pfleeger, C. Security in Computing. Prentice Hall, 1997.
[4] Mel, H.X. Baker, D. Cryptography Decrypted. Addison Wesley, 2001.
[7] Bryant, W. Designing an Authentication System: A Dialogue in Four Scenes.
[8] J. T. Kohl, B. C. Neuman, and T. Y. Ts'o. The evolution of the Kerberos authentication system. In *Distributed Open Systems*, pages 78-94. IEEE Computer Society Press, 1994.
[9] B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, *IEEE Communications*, 32(9): 33-38. September 1994.
[10] S.P. Miller, B. C. Neuman, J. I. Schiller, and J.H. Saltzer. Section E.2.1: *Kerberos Authentication and Authorization System*. Project Athena Technical Plan, MIT Project Athena, Cambridge, Massachusetts, October 1988. (Version 4)
[11] S. M. Bellovin and M. Merritt. Limitations of the Kerberos authentication system. *Computer Communication Review*, 20(5): 119-132, October 1990.

VIII. BIOGRAPHIES

Jitendra Singh post graduated M.C.A from IGNOU University and pursuing ALCCS equiv. to M.Tech. (CS) from IETE, New Delhi. His employment experience included the U.P. Technical University, Lucknow, CCS University, Meerut, Dr. B. R. Ambedkar University, Agra and presently working as a Sr. Lecturer in SRM-IMT, Modinagar Campus of SRM University Chennai. His special fields of interest included Network Security and User Authentication Mechanism

R. P. Mahapatra post graduated M.E (C.S.E) from University of Madras and pursuing PhD from Berhampur University Orissa. His employment experience included the Anna University, Madras University, Mekelle University, Ethiopia and presently working as an Assistant professor and HOD (CSE & IT) SRM-IMT, Modinagar Campus of SRM University Chennai. His special fields of interest included software engineering and network security.