Global Information Assurance Certification Paper

Single Sign On Concepts & Protocols

GSEC Practical Assignment Version 1.4b, Option 1

Author: Sandeep Singh Sandhu

Date: 30th January 2004

Abstract:
A single sign on infrastructure is increasingly becoming essential in modern enterprises with many users accessing multiple applications over large networks. This paper describes the characteristics and concepts of a few important protocols and technologies that have been used for implementing authentication and single sign-on (SSO) mechanisms for computer networks.

We begin by examining the main features expected of an SSO solution. The related problems of securely performing authentication and authorization in an insecure open network are explored. Some relevant concepts and protocols have been discussed in this context. These form the building blocks for designing a SSO infrastructure.

The Kerberos and Sesame protocols have been briefly described to illustrate the design of a full-fledged SSO architecture.

Table of Contents:

1 Introduction

A single Sign-on infrastructure provides transparent access to all network resources for a user with only a single login. It enables a user to access multiple computer platforms or application systems after being authenticated just one time [1]. The user's identity and authorization data is stored in this centralized setup (of 1 or more servers), which is trusted by all applications.
This has several advantages over having separate multiple logins to each network host [2]:

- Improved productivity since a user only needs to remember one SSO password to access every network resource or application.
- Easier and consistent administration of both users' and applications' security profiles.
- Simpler and more secure integration of security features during application programming since only standard library calls to the SSO API need to be called.
- Integration of security administration for disparate systems - applications running on different operating systems/hardware, etc.
- Improved network security - by implementing an SSO it is assured that passwords and sensitive data will be securely transmitted and managed for all applications. Users don't write down their multiple passwords to remember ensuring better security practices amongst users. A centralized profile administration to control and monitor user's access privileges.
- Lower cost of implementing and maintaining security across the enterprise. Security services & functionality need not be re-built from scratch for every new application. [3]

Single sign on can be implemented either as a common authentication/authorisation service with centralised identity management. This provides a common centralised infrastructure to which both users and hosts communicate to authenticate accesses to resources – e.g. Kerberos and Sesame.

Alternatively Single sign on can be implemented as a password synchronisation protocol. This type of implementation uses services/agents on all hosts to distribute and synchronise a user's password and/or access controls on all hosts whenever a user's profile is modified. While this does not provide benefit of centralised profile administration, it provides an easier and more cost effective means to implement SSO since architectural changes are not required for all hosts.

The implementation of a password synchronisation solution has been described in [4] at the SANS reading room. This approach is not discussed in the paper.

2 Features

An SSO must securely identify a user and support a number of different identity verification methods – password, dynamic passwords, hardware tokens, smart cards, digital certificates, biometric identifiers, etc. The user's access control information and the resource's access control profile must be stored centrally by the SSO for better security and administration. All authorisation and authentication messages and decisions must be secured when being transmitted on the network

from the SSO infrastructure to/from the host applications and users. All profile and security administration activity must be auditable and controlled securely.

The features required of an SSO infrastructure are: [5][2]

- The solution must be scalable so it can expand to serve the requirements of a large enterprise.
- An SSO infrastructure must be reliable and provide a fail-over arrangement
- An SSO should be able to support mutual authentication of client and server
- It should provide transmission level security (e.g. TLS or IPSEC)
- An SSO should guarantee integrity of data or confidentiality or both
- An SSO must provide sufficient flexibility to carry service specific attributes in messages
- Use reliable transport mechanisms for transmissions
- It should be able to support access certificates, or access control information, access rules, restriction filters.
- Its functions/operations should be auditable (e.g. audit trails/logs with timestamps for important events)
- The SSO infrastructure should be supported for operation in diverse environments on clients & servers running on different applications, operating systems, hardware, etc.
- It should be able to re-authenticate a user on demand, send updated authentication information and support user login time-outs.
- The authentication and access authorisation should be extendable to clients-servers across domain in a multiple domain/realm environment using the principles of propagation of trust by trust-chaining/trust proxy/hierarchy, etc.

SSO protocols must be protect against some of common types of attacks such as:

- Network sniffing for shared secrets
- Replay attacks
- Negotiating a weak authentication scheme when multiple options exist.
- Online dictionary attacks
- Man in the middle attacks (passive & active)
- Chosen plaintext attacks
- Pre-computed dictionary attack (pre-compute hashed values from a dictionary)
- Brute force attacks
- Spoofing by counterfeit servers
- Storage of authentication information by network cache's (proxy/gateway)

3 Concepts & Protocols

A few important concepts used throughout this discussion of SSO are explained in this section.

3.1 A Cryptography Primer

We introduce, very briefly, some important concepts related to cryptography which will be useful in understanding the protocols and practices discussed later. The topics are not all-inclusive of this vast subject; these are only intended to highlight some important cryptographic concepts used to cryptographically verify identity and trust. The reader is advised to refer to the excellent sources listed in the references section for a formal and complete description of these concepts [6][25].

3.1.1 Conventional encryption/Symmetric key or shared secret Encryption [6]

This type of encryption operation transforms plain text into cipher text (encrypted messages) using a secret data (key) known to the sender. This cipher message can now be safely transmitted over an unsecure channel of communication (e.g. internet) to the recipient. The recipient will be able to decrypt and read the message by using the same secret key used by the sender.

Its disadvantage is that the secret key needs to be sent securely to the recipient on a separate secure channel (out-of-band transmission e.g. password sent by post). Also, if the messages needs to be exchanged in this manner with a large number of users each pair of users would need a separate shared secret, e.g. to exchange messages between 10 users (10 x 9) / 2 = 45 shared secret keys are required. For *n* users, *n* x (*n*-1) ÷ 2 keys are required which becomes impractical to be implemented manually.

The strength of an encryption is measured in terms of the length of the secret key; this is specified in bits: 64, 128, 256, 512 bits. Keys should be large enough so that brute force attacks (guessing all possible key combination becomes computationally impractical. Many algorithms have evolved for encrypting data using symmetric keys – a few important ones used extensively are CAST128, TripleDES, AES Rjindeal, and IDEA.

3.1.2 Hash Functions and Message Authentication Codes (MAC/HMAC)[6]

A variety of methods have been designed for detecting errors and changes in data transmission. Even-odd parity checks have been used for error detection in serial data transfers or RAID hard disc arrays. 16 bit and 32 bit checksums are used for detecting media storage/retrieval errors for magnetic media/tapes/discs. Cryptographic hash codes/digests and message authentication codes (MAC) have similarly evolved for ensuring that the data being examined has not been tampered.

These are one-way functions whose output cannot be used to recover the original data. They generate a small fixed length code from a larger variable length message. The recipient can compare this code to that computed at his end using the message received earlier, and determine that the message was tampered if these are not equal. If the MAC is encrypted with a shared secret key or the user's private key, it can be sent along with the message itself on the same unsecure channel for immediate verification of integrity.

Another use of MAC codes is for encrypting a user's password and then send the MAC code over the unsecured channel or store it on the local file system. This protects the user's password from tampering or misuse. These mechanisms are discussed later in CHAP and Kerberos protocols.

Some of the commonly used hash/MAC codes are MD5, RIPEMD and SHA1, these can be of lengths of 128, 160, 256, or 512 bits. A larger size MAC is more resistant

to birthday attacks (comparing two messages to find a pair whose MAC codes match). This is also known as collision resistance of the MAC. To further strengthen MAC codes an HMAC mechanism has been proposed in RFC 2104 [7]. It enhances security by using a secret key along with the message as input to the hash function. This is described as:

```
      HMAC = Hash( Key XOR opad, Hash( Key XOR ipad, text ) )
Where,    Hash( A, B )  = the hash function applied over A and B
          Key = the secret key used in the HMAC operation
          ipad = byte 0x36 repeated 64 times
          opad = byte 0x56 repeated 64 times
          Text = message/data
```

3.1.3 Digital certificates and Public Key Encryption

In public key encryption two keys are created for each user – a private key to be kept secret and a public key to be distributed publicly. To encrypt a message the user uses the recipient's public key, encrypts and sends the data to the recipient. The recipient can decrypt the message using his private key.

Message integrity and non-repudiation can be assured by signing the message data. The sender can do this by encrypting the message with the sender's private key to create a message, which can only be decrypted by using the sender's public key. This validates that the message was only sent by the sender and has not been tampered after it was digitally signed. A combination of encryption and signing can ensure privacy, authenticity, integrity and non-repudiation of any message.

It has the advantage that for 10 users to exchange messages only 10 pairs of keys are required (or, *2n*) as opposed to 45 keys (or, *n x (n-1) / 2*) for symmetric encryption.

Public keys eliminate the problem of securely exchanging keys over an insecure channel. Public keys can be distributed on the same unsecured communication channel. A pubic key infrastructure is required to administer and authenticate the identities and validity of the keys used for public-private key encryption. A trusted third party Certificate authority signs each user's public keys to confirm the identity of that key. Alternatively, a web of trust model can be used where user's sign keys for people they trust and this extends to assure other users that the key signed by a person they trust can also be trusted. This approach has been used for Pretty Good Privacy (http://www.pgpi.org/).

3.1.4 Oakley protocol for key management [8]

The Oakley protocol provides a scalable and secure mechanism for key distribution on the Internet. The Diffie-Hellman key exchange [6] mechanism provides a secure way for two entities to agree on a secret key without requiring it to be transmitted on the unsecure channel. The STS mechanism demonstrates a way to embed this in a secure protocol to encrypt data and additionally validate each other's identities. The Oakley protocol adds several features to enhance these protocols:

- Providing a weak form of address validation by using anti-clogging tokens (cookies) to prevent Denial of Service attacks.
- The two parties can negotiate the encryption, authentication and key-derivation algorithms while setting up a secure channel.

- The keys are derived from an encryption algorithm in addition to the Diffie-Hellman mechanism. This strengthens the protocol.
- The protocol allows user-selectable and user-defined groups (mathematical structures) used in the encryption to be specified for performing the Diffie-Hellman key exchange.
- It also permits the use of authentication based on symmetric encryption or non-encryption algorithms. This flexibility is included in order to allow the parties to use the features that are best suited to their security and performance requirements

3.1.5 The S/KEY One-Time Password System [9]

An S/KEY system client passes the user's secret pass-phrase through multiple applications of a secure hash function to produce a one-time password. On each use, one reduces the number of applications. Thus a unique sequence of passwords is generated. The S/KEY system host verifies the one-time password by making one pass though the secure hash function and comparing the result with the previous one- time password.

The client's secret pass phrase should be more than eight characters; this is concatenated with a "*seed*" that is transmitted from the server in clear text. This non-secret seed allows a client to use the same secret pass phrase on multiple machines (using different seeds) and to safely recycle secret passwords by changing the seed.

The user's secret pass-phrase never crosses the network at any time, including during login or when executing other commands requiring authentication such as the UNIX commands "passwd" or "su". Thus, it is not vulnerable to eavesdropping/replay attacks. Added security is provided by the property that no secret information need be stored on any system, including the host being protected.

Operation:
A function on the host system that requires S/KEY authentication is expected to issue an S/KEY challenge. This challenge give a client the current S/KEY parameters - the sequence number and seed. The format of a challenge is:

```
s/key sequence_integer seed
```

The client can compute from a hardware device (or lookup a printed table) the one time password. It then passes it to the host system where it can be verified from a table or file storing the last successful login, or it may be initialized with the first one-time password of the sequence using the keyinit command.

3.2 Authentication & Authorisation

Authentication is the process of verifying a users identity [1]. An authentication process consists of two steps:

- **Identification**: Presenting an identifier to the security system (enter username)
- **Verification**: Presenting or generating authentication information that corroborates the binding between the entity and the identifier (password entry and verification)

The verification process involves validating the authentication information used to verify an identity claimed by or for an entity. It may be derived from:

- Something the entity knows. (e.g. a static password).

- Something the entity possesses. (e.g. a dynamic/one-time password generated from a hardware device/token, digital certificate)
- Something the entity is. (e.g. biometric authentication of fingerprint/palm geometry)

In an unsecure, open network, vulnerable to an active man-in-the-middle attack, it might be essential to assure the user that the host system is genuine and an authentication of the host is included as part of the authentication process. This might be done by displaying the user's profile information after login (last login time, etc) or validating the Host system's digital certificate signed by a trusted third party as described in the TLS protocol.

3.2.1 PPP authentication protocols

The Point-to-Point Protocol (PPP) provides a standard method for transporting multi-protocol datagrams over point-to-point links (defined in RFC1661). It consists of a link control protocol, which utilises the Password Authentication Protocol (PAP), the Challenge Handshake Authentication Protocol (CHAP) or the extensible authentication protocol (EAP) commands to configure a link.
These authentication protocols are intended for use primarily by hosts and routers that connect to a PPP network server, but can be applied to other scenarios as well.

Password Authentication Protocol (PAP) [10]

When the password authentication protocol is used in PPP after the link establishment phase is complete, an Id/Password pair is repeatedly sent by the client to the server until authentication is acknowledged or the connection is terminated. PAP has the disadvantages that passwords are sent "in the clear", and there is no protection from playback or repeated trial and error attacks. The client is in control of the frequency and timing of the attempts.

Challenge handshake authentication protocol (CHAP) [11]

The Challenge handshake authentication protocol involves sending a random Challenge to the user (client) to which the client responds with a cryptographically hashed response which depends upon the Challenge and a secret key. This value is then checked against the hashed value generated by the server from the shared secret key (e.g. password) and if equal, the authentication is successful and is acknowledged.

Its advantages are that:

- Shared secret is never sent over the network
- The challenge depends on an incrementally changing random value so the server is protected against a replay attack.
- The use of repeated challenges is intended to limit the time of exposure to any single attack. The server is in control of the frequency and timing of the challenges
- It can be used for mutual authentication if a client needs to authenticate a server.

The disadvantage of using this mechanism is that for evaluating the response hash, a server requires the shared secret to be available in plaintext form. Most password databases or files store shared secrets after irreversibly encrypting them (to protect against their misuse), hence these cannot be used for CHAP.

The challenge values must be unique and unpredictable to protect against replay attacks.

Extensible Authentication Protocols (EAP) [12]

PPP was not defined to be flexible enough to extend the types of authentication methods/commands hence the EAP was proposed as a non-negotiable authentication configuration during the link control protocol to specify the following additional authentication methods to be used:

- Identity
- Notification
- Nak (Response only)
- MD5-Challenge
- One-Time Password (OTP)
- Generic Token Card

These protocols are not directly related to application authentication methods required for an SSO but they do provide a valuable insight into how different protocols have evolved to securely authenticate users.

3.2.2 Mail Authentication Other Protocols: APOP, AUTH, and SASL

These protocols provide other techniques to auth users without sending a shared secret on insecure network.

POP3 APOP command [13]

The post office protocol supports the optional use of the APOP command to securely authenticate an email reader without sending the password in clear text on the network.

The command is issued as:

```
APOP name digest
```

Where the arguments are the user's id and a MD5 digest string. A POP3 server which implements the APOP command will include a timestamp in its banner greeting - the syntax of the timestamp might be: `<process-ID.clock@hostname>`. The POP3 client uses this timestamp for the APOP command. The `*digest'* parameter (16-octet value) is calculated by applying the MD5 algorithm to a timestamp string followed by the password.

The server verifies the digest provided. If the digest is correct, the POP3 server issues a positive response, and the POP3 session enters the TRANSACTION state, else a negative response is issued. As such, shared secrets should be long strings (considerably longer than the 8-character example shown below).

An APOP Example [13]

```
S: +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C: APOP mrose c4c9334bac560ecc979e58001b3e22fb
S: +OK maildrop has 1 message (369 octets)
```

In this example, the shared secret is the string `tanstaaf'. The MD5 algorithm is applied to the string `<1896.697170952@dbc.mtview.ca.us>tanstaaf` which produces a MD5 digest value of `c4c9334bac560ecc979e58001b3e22fb`

IMAP auth command [14]

The Internet Message Access Protocol, Version 4 contains the AUTHENTICATE command, for identifying and authenticating a user to an IMAP4 server and for optionally negotiating a protection mechanism for subsequent protocol interactions. The following authentication mechanisms can be used by this command:

- Kerberos_v4
- GSSAPI
- SKEY

A full description of the implementation is out of the scope of this paper.

Simple Authentication and Security Layer (SASL) Protocol: [15]

The RFC 2222 specifies a SASL mechanism for connection oriented protocols to identify and authenticate a user and optionally negotiate protection of subsequent protocol interactions for inserting a security layer between the connection and the protocol. If a server supports the requested mechanism, it initiates an authentication protocol exchange consisting of a series of server challenges and client responses that are specific to the requested mechanism. If the use of a security layer is agreed upon, then the mechanism must also define or negotiate the maximum cipher-text buffer size that each side is able to receive.

Other SASL extensions include RFC 2444 - “One Time Password SASL Mechanism” by Newman, RFC 2245 - “Anonymous SASL mechanism”, and RFC2831 - “Using Digest Authentication as a SASL Mechanism” by Leach & Newman.
SASL mechanisms have been defined for many protocols - Kerberos, GSS API, and S/key; other registered mechanisms are available at the IANA website at http://www.iana.org/assignments/sasl-mechanisms . SASL has also been used implemented with SMTP for securing mail server access.

3.2.3 HTTP Auth [16]

Since a large no of applications developed nowadays tend to be web-based clients, we examine in some detail the authentications protocols used for HTTP protocol.

HTTP provides a simple challenge-response authentication. It uses an extensible, case-insensitive token to identify the authentication scheme, followed by a comma-separated list of attribute-value pairs which carry the parameters necessary for achieving authentication via that scheme.

```
auth-scheme = token
auth-param  = token "=" ( token | quoted-string )
```

The following parameters are supported:

- **Realm**: It defines the protection space and allows the protected resources on a server to be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database. This token is given as: "`realm = <realm-value>`”.
- **Challenge**: This is sent as part of an unauthorised access error message 401, include a `WWW-Authenticate` header field containing at least one. The token is given as:

```
challenge   = <auth-scheme> <one or more #auth-param>
```

- **Credentials**: The Authorization field value consists of credentials containing the authentication information of the client for the realm of the resource being requested. The web-browser must choose to use one of the challenges with the strongest auth-scheme it understands and request credentials from the user based upon that challenge. The token is given as: "`credentials = <auth-scheme> #auth-param`"

The authentication scheme could be either "**basic**" or "**digest**" authentication.

Basic Authentication Scheme

This authentication scheme sends the password un-encrypted and only disguised in Base64 encoding character string. The unauthorised response (error 401) from the web server is given as:

```
WWW-Authenticate: Basic realm="My_Realm"
```

Where, "`My_Realm` " is the realm. To receive authorization, the client sends the *userid* and *password*, separated by a single colon (":") character within a base-64 [7] encoded string in the credentials.

```
basic-credentials = base64-user-pass
base64-user-pass  = <base64 encoding of user-pass>
```

Where, `user-pass = userid ":" password`

A Basic Auth Example [16]:

If the user agent wishes to send the user id "Aladdin" and password "open sesame", it would use the following header field in the response from the web browser:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Digest Authentication Scheme:

The server using this authentication scheme sends a challenge to the client containing a nonce value. The client responds back with a digest (default is 128 bit MD5 digest) of the username, the password, the given nonce value, the HTTP method, and the requested URI. This ensures that the password is never sent in the clear. The username and password must be prearranged for creating the digest in a fixed format.

The following tokens are sent by the server in a "401 Unauthorized" response, and a `WWW-Authenticate` header:

```
challenge       =  "Digest" digest-challenge
```

The digest-challenge is one or more of:

```
( realm | [ domain ] | nonce | [ opaque ] |[ stale ] | [algorithm] | [ qop-options ] | [auth-param] )
```

Where,

- **Realm** is the role or user space authorised to use the protected resource.
- **Domain** is the protected space being accessed by the client given as "`domain = <URI>`" where URI is the `absoluteURI` or `abs_path`
- **Nonce** is a base64 encoded or hexadecimal encoded string sent as "`nonce = <nonce-value>`". The server should prevent using or accepting a previously used nonce value to prevent replay attacks.

- **Opaque** represents a string to be returned by the client unchanged, this could be used for session tracking. The format is "`opaque = <quoted-string>`"
- **Stale** indicates the previous request had been rejected since the nonce was stale. The format of the token is "`stale = ( "true" | "false" )` "
- **Algorithm** string indicates a pair of algorithms used to produce the digest and a checksum given as "`algorithm= ( "MD5" | "MD5-sess" | token )`"
- **QOP-options** is an obsolete optional directive indicating the "quality of protection" for the connection, the format is: "`qop = auth | auth-int | token`"

The client responds with the following request header:

```
credentials = "Digest" digest-response
```

Here, the `digest-response` is one or more of the following separated with spaces:

```
(username | realm | nonce | digest-uri | response | [ algorithm ] |
[cnonce] |  [opaque] | [message-qop] | [nonce-count]  | [auth-param])
```

Where,

- The user's name given as "`username = username-value`"
- **Digest-uri** is the URI from Request-URI of the Request-Line; duplicated in the header because proxies are allowed to change the Request-Line in transit digest-uri, this is given as "`uri = <digest-uri-value>`", as specified by HTTP/1.1
- **Message-qop** is the Quality of protection of the message: "`qop = <qop-value>`"
- **Cnonce** is an opaque string sent only if the server sent a qpop value "cnonce=cnonce-value
- **Nonce-count** is the hexadecimal number of counts of requests sent by the client with nonces
- **Response** contains the authentication reply from the client given as: "`response = <request-digest>`". The request digest is calculated as:

```
"KeyedDigest(     (  Hash("username":"realm":"password") ,
                 nonce-value":"nonce-count":"cnonce-value":"qop-value":"
                 Hash("Method":"digest-uri-value")
                  )"
```

A Digest Auth Example [16]:

The first time the client requests the document, no Authorization header is sent, so the server responds with:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
      realm="testrealm@host.com",
      qop="auth,auth-int",
      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
      opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

The client may prompt the user for the username and password, after which it will respond with a new request, including the following Authorization header:

```
Authorization: Digest username="Mufasa",
      realm="testrealm@host.com",
      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
```

```
uri="/dir/index.html",
qop=auth,
nc=00000001,
cnonce="0a4f113b",
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

For some applications, TLS or SHTTP are more appropriate options.

3.2.4 Windows NTLMv2 protocol [17]

Before adopting Kerberos for Windows 2000, windows implemented the NT LanMan Version 2 (NTLMV2) challenge-response authentication protocol for operating system authentication for network resources- this is supported by Windows NT SP4 and Win 9x operating systems. It is an improved CHAP protocol which allows user's to get transparently authenticated to the domain controller, file servers, printers, or other Microsoft applications such as MS SQL database servers, MS Exchange mail servers and IIS web servers within the same domain. The mechanism works as follows:

- The user logs in, to which the server sends a challenge.
- A 16-byte hash (NTLM hash) is created by hashing the user's Unicode password using the MD4 function.
- The Unicode uppercase username is concatenated with the Unicode uppercase server name. This is hashed using MD5 HMAC function with the 16-byte NTLM hash as the key to create the 16-byte NTLMv2 hash.
- A block of data known as the "blob" is constructed as shown below [17]:

Bytes	Description	Content
0	Blob Signature	0x01010000
4	Reserved	long (0x00000000)
8	Timestamp	Little-endian, 64-bit signed value representing the number of tenths of a microsecond since January 1, 1601.
16	Client Challenge	8 bytes
24	Unknown	4 bytes
28	Target Information	Target Information block (from the Type 2 message).
(variable)	Unknown	4 bytes

- The auth server's local security authority (LSA) constructs a user ID (UID) which the user can use for all future sessions.

3.2.5 Transport Layer Security (TLS) [18]

Transport Layer Security specifies a protocol used to provide secure, encrypted, application independent socket communications for any application utilising TCP/IP.

TLS comprises of the following 2 protocols:

- TLS Record Protocol: It ensures that the data transmitted is private (encrypted using symmetric keys) and reliable (integrity check using HMAC/MAC hash codes). Other socket-based protocols are encapsulated by this protocol. It is a

layered protocol. At each layer, messages may include fields for length, description, and content.

The Record Protocol takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the result. Received data is decrypted, verified, decompressed, reassembled, and then delivered to higher-level clients.

- TLS Handshake Protocol: This protocol is encapsulated in the record protocol and allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security. It provides the following security functions:
    - Authentication of the client/server's identity using RSA, DSS, DSA, etc.
    - Secure negotiation of the shared secret.
    - Reliable & tamper proof negotiation of the secrets

The protocol has been based on the SSL ver 3 designed by Netscape initially for HTTPS communications.

The following record-protocols are described for TLS version 1:

- TLS Handshake protocol
- Alert message protocol: Alert messages convey the severity of the message and a description of the alert. Alert messages with a level of fatal result in the immediate termination of the connection.
- Change cipher spec protocol: It consists of a single message sent by both the client and server to notify the other party that subsequent records will be protected under the newly negotiated CipherSpec and keys.
- Application data protocol

The record layer of the protocol performs the following functions:

- Fragmentation of the received variable length plaintext into data chunks of 2^{14} bytes or smaller length.
- All records are compressed if a compression algorithm is specified.
- Payload protection functions encrypt the data using stream ciphers or block ciphers A MAC code to detect the integrity of the data, the Mac record also contains a sequence number to detect repeated messages.
- Key generation: The Record Protocol generates keys, IVs, and MAC secrets from the master key provided by the handshake protocol. The master secret is hashed into a sequence of secure bytes, which are assigned to the MAC secrets, keys, and non-export IVs required by the current connection state.

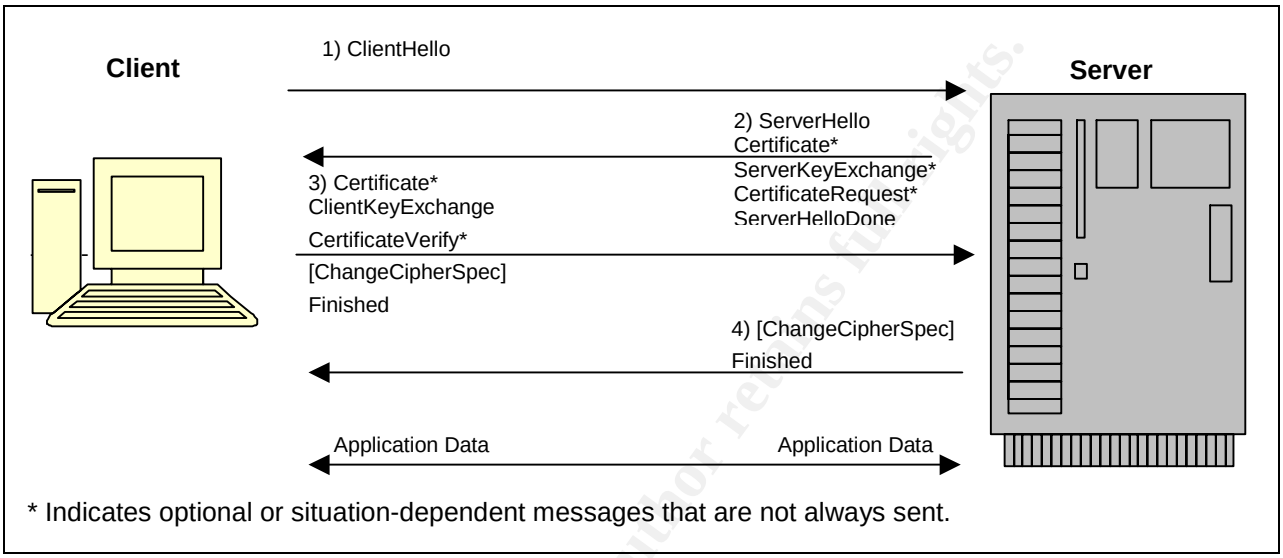
The TLS handshake protocol consists of a set of 3 sub-protocols - the change cipher specs protocol, alert messages protocol, and the application data protocol.

The handshake consists of the following steps:

- Send initial "hello" messages to agree on algorithms, exchange random values, and check for session resumption.

- Pre-master secret key agreement
- Exchange certificates and cryptographic information to allow the client and server to authenticate themselves.
- Generation of a master secret key from the pre-master secret key and exchanged random values.
- Provide security parameters to the record layer.
- Verification of security parameters and integrity check of the handshake.

Figure 1: Message flow for a full TLS handshake



3.2.6 Smart Cards[19]

Smart cards provide an alternative to entering passwords for confirming identity. They provide the authentication of the type "something you have". These cards contain a microprocessor chip that stores the user's secret key/private key. This is accessed from a smart card reader terminal/interface which then utilises the secret key/data for user authentication and encryption. The key might also optionally be protected by a user password to access the key from the chip. This access control may also be further used to supply the private key used in public key encryption, or it may be coupled with a biometric id system to further increase the number of tokens used in authentication.

3.2.7 Biometric Authentication

Biometric authentication utilises the principle of identifying an entity on the basis of "something the person is". Devices and technologies exist to utilise the following aspects of uniqueness of an individual for authentication [19]:

- Fingerprint scan
- Retinal blood vessel scan
- Iris pattern scan
- Palm geometry
- Voice pattern recognition
- Facial thermography

The functions performed by a Biometric system are getting data from the hardware interface, message transmission/communication to the authentication servers, signal processing and match decision-making.

The protocols for universally utilising biometric information from such devices and their use in authentication systems are being developed and standardised. One significant standard is X9.84. The BioAPI specification targets to create an platform independent universal API for vendor-independent utilisation of such devices in computer systems (BioAPI ver 1.1 is the ANSI standard - ANSI/INCITS 358-2002). Products implementing the BioAPI are listed at http://www.bioapi.org/BioAPI_products/products.htm

3.2.8 Access control information storage

The access control information describing the various users, applications, files, printers, and other resources accessible from a network is often collected into a special storage protected storage to be used in verifying and making access control decisions.

These can be stored in various ways:

a) LDAP Directory Server & Protocol [20]

The object (resources) and subject (users) information is sometimes stored in a directory, which is a type of a hierarchal database. Directories allow users or applications to find resources that have the characteristics needed for a particular task. Directories are different from databases in that they are read much more than written and are optimized for high volumes of read requests. Write access might be limited to system administrators or to the owner of each piece of information.

A client application that wants to read or write information in a directory calls a function or application programming interface (API) that causes a message to be sent to a process on the server which accesses this information. When a directory is distributed (and not centralized), the information stored in the directory can be partitioned or replicated on several directory servers. [20]

LDAP defines a standard method for accessing and updating information in a directory. It derives its design from the DAP (directory access protocol) developed to access the hierarchal namespace structure directory defined by CCITT in the X.500 standard (created in 1988, which became ISO 9594). The latest LDAP version 3 is defined in RFC 2251.

LDAP defines the content of messages exchanged between an LDAP client and an LDAP server. The messages specify the operations requested by the client (search, modify, delete, TCP/IP session operations, etc.), the responses from the server, and the message formats.

In LDAP, a directory entry describes an object, which is identified by a “Distinguished name” (DN) and can contain multiple attributes with types and values. These are organized in a tree structure called the Director information tree (DIT). The object is described by its class, classes are stored as part of a schema.

The following operations are defined for LDAP access:

- Query: search and compare data for objects/attributes
- Update: Add, delete, modify RDNS and directory tree

- Authentication: Bind (initiate and authenticate a session to the server using Kerberos/SASL/etc.), unbind (terminate session) and abandon.

A LDIF file format is defined to convey directory information as a series of records separated by line separators. A record consists of a sequence of lines describing a directory entry or a sequence of lines describing a set of changes to a single directory entry. An LDIF file specifies either a set of directory entries or a set of changes to be applied to directory entries. A sample entry is given below [20]:

```
dn: cn=John Smith, ou=people, o=ibm.com
objectclass: top
objectclass: organizationalPerson
cn: John Smith
sn: Smith
givenname: John
uid: jsmith
ou: Marketing
ou: people
telephonenumber: 123-4567
```

LDAP over TLS: To secure all access control information and communications, the LDAP access can be secured using TLS protocol. This secures all profile accesses and administration activities, ensure confidentiality, and can verify the identity of the hosts/initiators based TLS authentication methods (including client digital certificates).

A useful case study on implementing LDAP based access control has been described in a paper by Andres Andreu [21]. Another useful guide by Ellen Smith on implementing LDAP is available at [22]

b) File System Based Access Control

NTFS[23] contains access control lists for controlling access to the objects stored on the file system and the registry. The MFT (master file table) record for every file and directory on an NTFS volume contains a security descriptor (SD) attribute which contains information related to security and permissions for the corresponding object.

Every object's SD contains 2 access control lists (ACLs):

- System access control list (managed/used by the system)
- Discretionary access control list (store permissions for users and groups)

Each ACL has access control entries (ACE), which contains an ID to identify a user/group and the set of permissions applicable for the user/group to the object.

ACLs are also interpreted as per its inheritance model – static (default ACL of parent directory) or dynamic (changes with parent ACL change).

Unix file systems (such as ext2, ext3, reiserfs, UFS) also store information such as file read, write or execute permissions. They also support features such as set-user-id flags and set-group-id flags on executable files/programs to propagate trust.

c) Database storage of Access control information

The user's access profile information can be stored in a generic database as tables and relationships for storing identification information and authorisation information.

For example: Tables for user names and encrypted passwords and roles/groups for users can be used to provide a convenient storage and retrieval of access control information.

The advantage of using generic databases is that programming language and API interfaces are readily available to access the information from databases – this makes implementing such a solution very attractive.

3.2.9 Security Assertion Mark-up Language (SAML) [24]

The Security assertion mark-up language (SAML) is an XML message format that defines a protocol specification to use when two servers need to share authentication information. The protocol uses the web infrastructure where XML data moves over HTTP protocols on TCP/IP networks.

With SAML, any point in the network can assert that it knows the identity of a user or piece of data. It is up to the receiving application to accept if it trusts the assertion. SAML provides a mutually agreed-upon mechanism that states the trust level for any given user or group by specifying how to represent users, identifies what data needs to be transferred, and defines the process for sending and receiving authorization data.

The SAML specification also provides insight into the design issues for building an interoperable, Web-enabled system.

A few products which implement SAML are: [24]

- IBM Tivoli Access Manager
- Oblix NetPoint
- SunONE Identity Server
- Baltimore, SelectAccess
- Entegrity Solutions AssureAccess
- Internet2 OpenSAML
- Netegrity SiteMinder
- Sigaba Secure Messaging Solutions
- RSA Security ClearTrust
- VeriSign Trust Integration Toolkit
- Entrust GetAccess 7

The latest version of the SAML specifications (ver 1.1) can be found at the OASIS site at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.

3.2.10 The Generic Security Service API Mechanism (GSS-API) [26]

The Internet RFC 1508 defines GSS-API as an API which provides security services to callers in a generic fashion, supportable with a range of underlying mechanisms and technologies and hence allowing source-level portability of applications to different environments. The specification defines GSS-API services and primitives at a level independent of underlying mechanism and programming language environment, and is to be complemented by other, related specifications.

GSS aims to provide a high-level abstraction layer on top of different low-level security services allowing programmers to develop applications utilizing the SSO architecture

Applications, which require use of the API, call on GSS-API functions in order to protect its communications with authentication, integrity, and/or confidentiality security services.

A GSS-API caller accepts tokens provided to it by its local GSS-API implementation and transfers the tokens to a peer on a remote system; that peer passes the received tokens to its local GSS-API implementation for processing.

The security services available through GSS-API in this fashion are implementable over a range of underlying mechanisms based on secret-key and public-key cryptographic technologies.

GSS-API comprises of the following elements:

- Credentials - a structure for establishing a peer's security context related information.
- Tokens: These are data elements used in GSS-API calls, they can be context-level or per-message tokens.
- Security contexts
- Mechanism types – these specify the common mechanism selected by both peers while establishing the security contexts.
- Naming of structures
- Channel Bindings

A Java reference implementation has been provided by Sun Microsystems and is distributed as part of the JDK1.4 libraries. A useful guide for utilising the GSS-API functions in java programs has been given by Faheem Khan [27]. It demonstrates the ease with which programmers can use the GSS-API for integrating features such as Kerberos authentication into their programs using the Java GSS-API.

Windows 2000/2003 provides a Microsoft implementation of GSS-API called the SSPI (security service provider interface), which provides a Microsoft API for implementing the interfaces specified by GSS-API. It allows security service providers (SSPs) to provide their security implementations according to a standard interface defined for the Microsoft Windows operating system.

Many commercial and open-source C-language implementations are available for GSS-API. Entrust provides an entrust implemented GSS-API compliant API via its Entrust Authority Toolkit in C language for GSS-API available at http://www.entrust.com/authority/gss_api/features.htm. It implements the SPKM (Simple public key mechanism) version 1 and 2.

4 SSO Protocols

4.1 The Kerberos Network Authentication Service (Ver5) [2] [28]

The Kerberos service provides a trusted third party authentication service in which each entity or principle (server or client) is a user to the Kerberos service and believe Kerberos' judgment of its peer's identity.

Kerberos uses shared secret key symmetric encryption. For a user the secret is the user's password applied to a one-way function. Kerberos uses Propagation mode CBC DES (cipher block chaining) encryption which propagates an error throughout the message if it is changed/tampered/damaged.

Kerberos keeps a database of the users and their secret keys. The secret keys are negotiated at registration. It also generates temporary session keys which are only used for one session. Kerberos can be configured to use only authentication features, or integrity protection, or both. Collision-resistant checksums (SHA1/MD5/etc.) are generated and used to protect the integrity of messages.
It utilizes libraries for providing encryption, database administration, authentication, administration, database replication, and user/application programs.
In the Kerberos database records are held for each principal, containing the name, secret key, expiration date and other administrative data. Names are of the format `name.instance@realm`, where - the primary name is the name of the user or service, instance is the name of the machine where the server runs, and realm is the administrative domain that maintains the authentication data. There is one copy of the master database/server on which user administration can be done, but there can exist several slaves which provide a read-only database for authentication. The database can be replicated on these read-only slave machines, which receive periodic updates from the master machine.

There are 2 types of credentials used in Kerberos: tickets (used to securely pass user's identity between the auth server and application) and authenticators (additional user information to prove the client's identity). Tickets are secured with the server' secret key and can be used by the user several times during a session and is described as:

$$Ticket = Encrypt_{serverKey}(serverid, clientid, client\ network\ address, timestamp, ticket\ lifetime, client\text{-}server\ sessionKey)$$

An authenticator can only be used once during authentication, it is given as:

$$Ticket = Encrypt_{sessionKey}(\ clientid, client\ network\ address, timestamp)$$

A principal needs to perform the following function to use Kerberos authentication:

- Client sends a request for a *Ticket-Granting-Server* (TGS) ticket to the Kerberos server by sending the client name and TGS name.
- The Kerberos server authenticates the client and issues session keys for the TGS-client exchange, the TGS ticket encrypted using the TGS-Kerberos server keys; all encrypted using the client's key.
- Request for a service ticket from the TGS by sending the service name, TGS ticket and the client authenticator, both encrypted in the TGS-client session key.
- The TGS server checks for the clients access controls and issues a client-server session ticket encrypted using the server key and the client-server session key, both encrypted using the TGS-client session key. This is used in all server accesses made by the client to the same server.
- An optional mutual authentication may be done where the server sends the client the timestamp value+1 encrypted in the server-client session key.

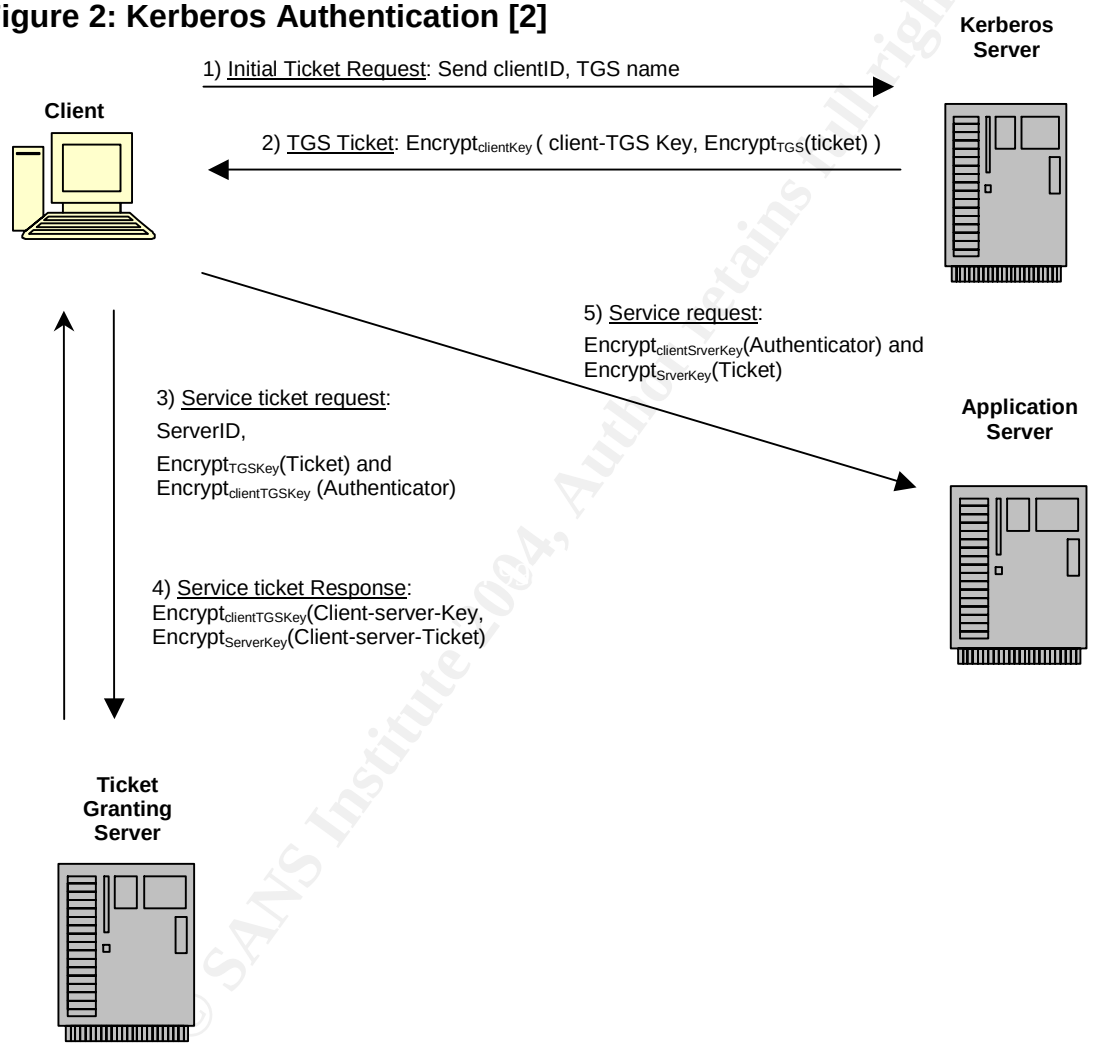
This process of Kerberos authentication is explained in figure 2. The other functions performed by Kerberos are:

- Administration functions: password change, add/delete principals, etc.
- Periodic Kerberos database replication onto the slave machines
- Interaction with other Kerberos realms for cross-realm trust propagation

A few Assumptions are made while implementing Kerberos [28]**:**

- Network is not protected against Denial of service attacks by Kerberos
- All principles must keep secret keys safe.
- The network must be secure against password guessing attacks
- All hosts on the network, which are part of Kerberos, must be loosely synchronized for timestamps to be effectively used.
- Principle names/ACL entries should not be recycled on a short-term basis to prevent identity theft.

Figure 2: Kerberos Authentication [2]



Implementing Kerberos: Kerberos implementation involve – setting up authentication modules for the OS/applications, installing the Kerberos server, creating a database and user accounts, realms, setup access to services, and setup trust and account mappings for foreign-domains.
There is a useful article on implementing Kerberos to integrate Windows 2000 and Linux OS authentication written by Alan Withers available at http://barney.gonzaga.edu/~awithers/integration/. Another good article on Kerberos

implementation by David Smith can be found at http://www.samag.com/documents/s=1769/sam0112d/0112d.htm.
Unix/Linux PAM modules (Pluggable authentication modules) for Kerberos are available at http://www.kernel.org/pub/linux/libs/pam/modules.html which enable quick integration of Kerberos authentication for a UNIX/Linux shell. A Kerberos authentication module for Apache "Mod_auth_kerb" is a module designed to provide Kerberos user authentication to the Apache web server (http://sourceforge.net/projects/modauthkerb/). It uses the Basic Auth mechanism to retrieve the username/password pair, and supports mutual authentication from some browsers via plugins/patches.

4.2 The SESAME Protocol v4 [29]

Secure European System for Applications in a Multi-vendor Environment (The SESAME Project)", is designed similar to Kerberos, but improves it by implementing concepts such as public key infrastructure for better secret key protection and key distribution.

Sesame supports access control, communications integrity and confidentiality while ensuring access control to services is controlled to appropriate level of security [29]. Sesame provides a core functionality over which vendors can build their products. For example, the Open Software Foundation's Distributed Computing Environment (DCE) has merged basic Kerberos functionality with sesame architecture concepts. The GSS API interface is also implemented in sesame, which hides implementation specific distributed authentication and access control details.

SESAME supports role based attributes for role-based access controls, hence builds authorization along with authentication services. It also supports the concept of delegation of a user's privileges to another user/application. To fully utilize the public-key-infrastructure, iinterfaces are defined for online or offline access to certification authorities(CA) or registration authorities(RA). A Key distribution server can optionally be used to perform key mediation during the authentication process; this server is used to deliver basic keys, manage long-term secret keys with services, manage mappings of services with names/domains, and support inter-domain operations.

Since SESAME uses PKI, its principles/entities are given DNs (distinguished names similar to DNs used in LDAP). Symmetric key encryption, HMAC algorithms and asymmetric key (public key) algorithms are used by SESAME. Two types of session keys have been defined – basic key (protect PAC integrity) and dialog keys (protect other data exchanges).
The process for authenticating a user in SESAME is described in brief:

- The users (initiator) logs onto the central authentication server of the sesame infrastructure.
- After successful authentication, the initiator is given a ticket that he presents to a privilege attributes server (PAS) to get a privilege attributes certificate (PAC). This access control certificate provides proof of his access rights and is signed by the public key of the privilege attribute server.
- The initiator provides this access control certificate to a target application when access to its protected resource is required. The initiator is given keying

information once it has selected an application to access. One part of this protects the certificate while another part protects the integrity and confidentiality of the data exchanged between the initiator and the target application. The actual key used is either constructed by the initiator itself, or by a key distribution server.

5 Conclusion

We have examined some important protocols which are designed to securely authenticate and authorise network users as SSO and keep data secure communications. Modern Cryptographic algorithms offer numerous possibilities for system designers to develop newer and better versions of the protocols described here. These protocols cater to a wide variety of application security requirements and may be implemented as such in newer areas for improving network security.

6 References:

[1] R. Shirey , "Internet security glossary", RFC2828 , May 2000 URL: http://www.ietf.org/rfc/rfc2828.txt, Jan 2004

[2] Jennifer G. Steiner - MIT, Cliffored Neumann, Jeffery I Schiller – MIT, "Kerberos: An authentication service for Open Network Systems", 30-mar-98, URL:http://secinf.net/uplarticle/2/kerberos.ps Jan04

[3] Schiener, "step-by-step instructions for implementing Kerberos on windows 2000", URL: http://www.microsoft.com/windows2000/techinfo/planning/security/kerbsteps.asp, Jan 2004

[4] Nancy Loveland "Single Sign On Through Password Synchronization" 6 Feb 2002 URL: http://www.sans.org/rr/papers/6/140.pdf Jan, 2004

[5] Network Working Group, RFC 2989, "Criteria for Evaluating AAA Protocols for Network Access" – 2000, URL: http://www.faqs.org/rfcs/rfc2989.html Jan 2004

[6] William Stallings, "Cryptography and network security" 2nd edition, 1999

[7] H.Krawczyk, M.Bellare, "HMAC", RFC2104, Feb 97, url: http://www.ietf.org/rfc/rfc2104.txt, Jan04

[8] H. Orman, "The Oakley Key Management Protocol" RFC 2412, November 1998, URL: http://www.ietf.org/rfc/rfc2412.txt, Jan 2004

[9] N. Haller, "S/Key Protocol", RFC1760, Feb95, URL: http://www.ietf.org/rfc/rfc1760.txt Jan 04

[10] B. Lloyd, W. Simpson,"PAP ", RFC1334,oct 92, url: http://www.ietf.org/rfc/rfc1334.txt, Jan 04

[11] W. Simpson, "CHAP", RFC1994, Aug 92, URL: http://www.ietf.org/rfc/rfc1994.txt, Jan 2004

[12] Blunk, L. and J. Vollbrecht, "PPP Extensible Authentication Protocol (EAP)", RFC 2284, March 1998; URL: http://www.ietf.org/rfc/rfc2284.txt, Jan 2004

[13] J. Myers - Carnegie Mellon, M. Rose - Dover Beach Consulting, Inc. "Post Office Protocol - Version 3", May 1996, URL: http://www.ietf.org/rfc/rfc1939.txt , Jan 2004

[14] J. Myers , "IMAP v4" RFC 1731, Dec 94, URL: http://www.ietf.org/rfc/rfc1731.txt , Jan 2004

[15] J. Myers , "SASL" RFC 2222, oct 97, URL: http://www.ietf.org/rfc/rfc2222.txt, Jan 2004

[16] Network working group,"HTTP Authentication",RFC2617, Jun99,url:http://www.ietf.org/rfc/rfc2617.txt , Jan 04

[17] Eric Glass, "The NTLM authentication protocol" 2003, URL: http://davenport.sourceforge.net/ntlm.html#theNtlmv2Response, Jan 2004

[18] T. Dierks - Certicom, C. Allen - Certicom, "The TLS Protocol", January 1999 URL: http://www.faqs.org/rfcs/rfc2246.html , Jan 04

[19] Micki Krause, Harold F. Tipton: "Handbook of Information security management" URL: http://www.cccure.org/Documents/HISM/ewtoc.html, Jan 2004

[20] IBM Redbook "Understanding LDAP", June 1998 URL: http://www.redbooks.ibm.com/redbooks/pdfs/sg244986.pdf, Jan 04

[21] Andres Andreu: “Using LDAP to solve one company's problem of uncontrolled user data and passwords” (SANS reading room), 30-Oct-2003,
URL: http://www.sans.org/rr/papers/9/1291.pdf, Jan 04
[22] [ELLE03] Securely implementing LDAP by Ellen Smith, 29-jul-2001,
URL: http://www.sans.org/rr/papers/6/109.pdf, Jan 2004
[23] [ntfs] Charles M. Kozierok - The PC Guide, Version: 2.2.0, “Access Control Lists (ACLs) and Access Control Entries (ACEs)” , April 17, 2001
URL: http://www.pcguide.com/ref/hdd/file/ntfs/secAccess-c.html, Jan 2004
[24] Frank Cohen “Debunking SAML myths and misunderstandings” - 8 July 2003, URL: http://www-106.ibm.com/developerworks/xml/library/x-samlmyth.html?Open&ca=daw-se-news, Jan 04
[25] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone “Handbook of Applied Cryptography” - October 1996
[26] RFC1508 J. Linn, Geer Zolot Associates, “Generic Security Service Application Program Interface”, September 1993, URL: http://www.ietf.org/rfc/rfc1508.txt, Jan 2004
[27] Faheem Khan, “Design secure client/server Java applications that use GSS-API and Kerberos tickets to implement SSO” (*9-Sep-03),*
URL: http://www-106.ibm.com/developerworks/java/library/j-gss-sso/
[28] J. Kohl , RFC1510, “Kerberos Network authentication service ver 5”, Sep 1993, URL: http://www.ietf.org/rfc/rfc1510.txt, Jan 04
[29] Tom Parker - ICL, Denis Pinkas – BULL, Issue 1, “SESAME Technology Version 4”, Dec-95, URL: http://www.isrc.qut.edu.au/sesame/doc-txt/overview.txt, Jan 2004